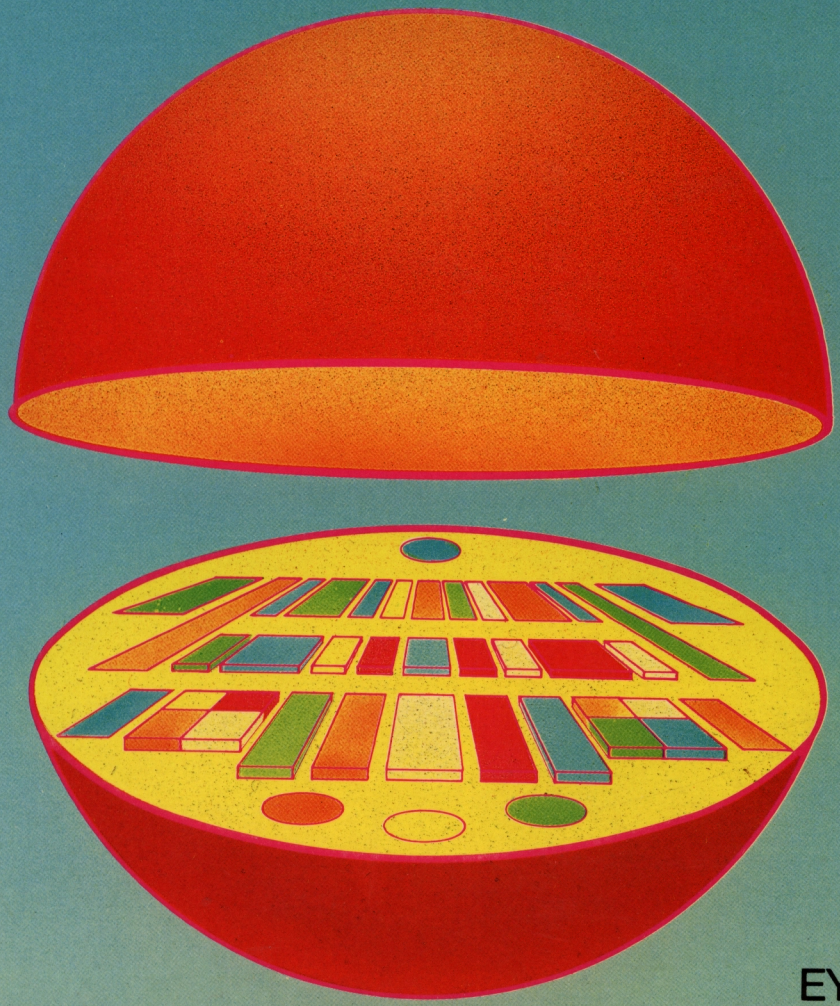
TOUT SAVOIR SUR

ORIC



EYROLLES

ROGER POLITIS . BRUNO VANRYB

TOUT SAVOIR SUR

ORIC

CHEZ LE MÊME ÉDITEUR

Dans la même collection

VANRYB-POLITIS	— *Tout savoir sur New-Brain*, 104 p. ; 1984.
GOLDSTEIN	— *Le guide de l'IBM-PC*, 272 p. ; 1984.
BUNN	— *Obtenez le maximum de votre ATARI*, 104 p., 1984.
GARCIA	— *Le Basic minimum.* 15 mots pour apprendre à programmer, 104 p., 1984.
WANNER	— *Aller plus loin en BASIC TO7*, 312 p., 1984.
POLITIS-VANRYB	— *Tout savoir sur ORIC*, 168 p., 1984.
VANRYB-POLITIS	— *Tout savoir sur Multitech*, 152 p., 1984.
DELANNOY	— *Faites vos jeux avec ORIC*, 224 p., 1984.
IFRAH	— *Faites vos jeux avec CANON XO7*, 104 p., 1984.
RAVEL	— *Tout savoir sur LASER 200-210*, 112 p., 1984.
VANRYB-POLITIS	— *Tout savoir sur LYNX*, 176 p., 1984.

COLLECTION MICROPLUS

TOUT SAVOIR SUR ORIC

par

Roger POLITIS
Bruno VANRYB

EYROLLES

61, boulevard Saint-Germain — 75005 PARIS
1984

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

«La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les «copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective» et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, «toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite» (alinéa 1er de l'article 40)».
«Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal».

© Éditions EYROLLES, 1984

Introduction

Ces dernières années, le marché de l'ordinateur individuel a été marqué par une affluence de petites machines performantes et économiques, souvent originaires de Grande-Bretagne. L'ORIC-1 s'inscrit avec bonheur dans cette lignée, dont il représente à la fois le dernier rejeton et une certaine forme d'aboutissement.

En effet, l'ORIC réunit à la fois toutes les caractéristiques qui font le succès d'un ordinateur "familial", couleur et son très développés, graphique performant avec d'autres beaucoup moins fréquentes dans cette gamme de prix: BASIC évolué d'où pratiquement rien ne manque, interface parallèle pour imprimante et possibilités de communications entre autres.

Ce livre se voudrait une extension du manuel ORIC, destiné à ceux qui, ayant assimilé les principes de base de la programmation, désirent mieux connaître et utiliser leur machine. Toutes les parties du manuel présentant une certaine difficulté ou traitant de fonctions peu évidentes sont évidemment développées à fond. Mais le lecteur trouvera aussi de nombreux "trucs" lui permettant de tirer un bien meilleur parti de son ORIC, ainsi que plusieurs véritables programmes d'applications.

La première partie, après avoir clarifié quelques notions indispensables, traitera du BASIC résident et en particulier de toutes les commandes et fonctions peu usitées dont l'ORIC est assez prolixe. Des petits programmes

et des astuces permettant d'obtenir les fonctions ou instructions absentes du BASIC complètent le texte, partout où cela est nécessaire.

La deuxième partie fait le point sur l'organisation mémoire de l'ORIC, la façon dont il "range" toutes les informations qu'on lui donne et comment elles sont exploitées. On y trouvera une carte détaillée de la mémoire et la liste des principales variables-système avec leur fonction, ainsi qu'un programme de renumérotation de lignes.

La troisième partie est consacrée à l'écran: L'utilisation des codes de contrôle et la gestion d'écran en modes TEXT et LORES y sont largement abordés. Tous les codes, très nombreux sur ORIC, permettant d'obtenir des fonctions spéciales aussi bien en ce qui concerne l'affichage que la couleur sont expliqués et illustrés en accès direct et en mode programme. Les extensions possibles en langage-machine sont aussi étudiées, comme les fonctions de l'éditeur d'écran. Vous trouverez dans ce chapitre, entre autres, un programme de recopie d'écran-texte sur imprimante et une sauvegarde d'écran sur cassette.

La manière de programmer ses propres caractères est également expliquée de façon exhaustive, illustrée par un grand programme de générateur de caractères intéractif, permettant de générer ses propres jeux de caractères aussi facilement qu'en les dessinant sur une feuille de papier.

Un programme de traitement de textes, offrant toutes les fonctions de saisie et d'édition indispensables et utilisant la plupart de ces commandes, termine cette section.

Le Graphique Haute Résolution est traité en quatrième partie. Toutes les instructions spéciales sont évidemment détaillées, en insistant plus particulièrement sur les moins évidentes comme FILL et PATTERN.

Un programme de D.A.O. (Dessin assisté par ordinateur), permettant d'exploiter toutes les possibilités graphiques de l'ORIC, et comportant entre autres une partie Hardcopy (copie d'écran sur imprimante), clôt ce chapitre.

La dernière section enfin est consacrée au son, qui, étant géré par un processeur spécialisé, fait l'objet d'un nombre relativement important d'instructions et confère à l'ORIC de réelles possibilités musicales. Celles-ci, en particulier l'instruction "PLAY", sont toutes examinées et illustrées d'un programme mettant en évidence les effets qu'il est possible d'obtenir.

Un jeu intéractif du type "jeux d'arcades", mettant en œuvre à la fois la création de caractères spéciaux et l'utilisation de bruitages et de musique, terminera ce chapitre et l'ouvrage.

Table des matières

Introduction VII

1. Le Basic ORIC 1

1.1. Introduction 1

1.2. Quelques rappels 2

1.2.1. Instructions et fonctions 2
1.2.2. Variables et constantes 3
1.2.3. Les opérateurs 4
1.2.4. Mode direct et mode programme 5
1.2.5. Pour accroître l'efficacité de vos programmes 5

1.3. Conventions d'écriture 6

1.4. Les instructions du Basic 8

1.5. Les fonctions du Basic 19

1.6. Insructions et fonctions non résidentes 26

1.6.1. Instructions 26
1.6.2. Fonctions 30

2. L'organisation mémoire et les périphériques 33

2.1. Généralités — Rappels 33

2.2. Le memory-map 34

2.3. Les différentes zones-mémoire 36

2.3.1. Les pointeurs-systèmes 0-#FF (0-255) 37
2.3.2. La pile #100-#1FF (256-511) 44
2.3.3. Les variables-système #200-#2FF (512-767) 45
2.3.4. Les autres zones de la mémoire 51

2.4. La gestion des périphériques 52

2.4.1. L'imprimante 53
2.4.2. Le magnétophone à cassette 54

3. L'écran et les jeux de caractère 61

3.1. La représentation de l'écran en mémoire 61

3.1.1. Ligne et colonnes réservées 63

3.2. Écrire avec Print — Les codes de contrôle 63

3.2.1. Les codes de contrôle 64
3.2.2. L'éditeur de ligne 66

3.3. Utilisation de Escape 68

3.3.1. La Hardcopy texte 70

3.4. Utilisation de Plot — Les attributs d'écran 71

3.4.1. Le programme "Tennis" 73

3.5. Extensions au Basic — Le langage machine 76

3.5.1. Écrire et lire l'écran avec POKE et PEEK 76
3.5.2. L'utilisation des variables-système 77

3.6. Modes texte et LORES 80

3.6.1. Le mode LORES 80
3.6.2. Choix d'attributs pour tout l'écran 80

3.7. Les jeux de caractères 81

3.7.1. Généralités 81
3.7.2. Principes de codage 82
3.7.3. Comment créer un caractère 83

3.8. Les programmes d'applications: CARGEN et GENETEX 84

3.8.1. Le programme "CARGEN" 84
3.8.2. Le programme "GENETEX" 91

4. Le graphique haute résolution 105

4.1. Introduction 105

4.2. Organisation-mémoire du graphique 106

4.3. Les commandes graphiques du Basic 108

4.3.1. Instructions 109
4.3.2. Fonction 114

4.4. Quelques programmes 114

4.5. Dessin Assisté par Ordinateur (D.A.O.): Le programme MULTIGRAF 119

5. Le son ORIC 130

5.1. Principes de fonctionnement 130

5.2. Commandes disponibles 131

5.3. Une application: le programme MONOSYNTH 134

5.4. N-BOMBER 138

Appendices 146

1. Le jeu de caractères ASCII 146

2. Index des adresses mémoire 148

3. Les messages d'erreur 150

Index alphabétique 153

1

Le Basic ORIC

1.1. Introduction

Ce chapitre n'a pas pour but de présenter in extenso toutes les commandes Basic, dont on peut trouver l'explication dans le Manuel. Nous nous étendrons plutôt sur toutes celles d'entre elles qui, soit parce qu'elles sont peu fréquentes, soit parce que leur utilisation n'est pas évidente au premier abord, nécessitent une définition plus détaillée. D'autre part, et malgré le fait qu'il possède un Basic particulièrement étendu, certaines commandes font défaut à l'ORIC. Chaque fois que cela a été possible, nous avons tenté de palier à cette absence par l'intermédiaire d'un petit programme.

1.2. *Quelques rappels*

Avant toutes choses, nous pensons qu'il est utile de préciser quelques notions de base, sur le fonctionnement des micro-ordinateurs, et celui d'ORIC en particulier.

ORIC utilise un langage évolué, le Basic, pour communiquer avec nous. Les ordres donnés en Basic sont ensuite interprétés et exécutés un à un. Le programme interne chargé de cette traduction s'appelle l'**interpréteur Basic**, et il a été écrit une fois pour toutes : On ne peut pas le modifier, et la mémoire le contenant s'appelle mémoire morte ou MEM en français, et ROM (Read Only Memory) en anglais. Le langage Basic est constitué d'un ensemble d'**Instructions**, de **Fonctions**, d'**Opérateurs** et d'**Opérandes** ou **Arguments** ; voici l'explication de ces termes :

1.2.1. Instructions et fonctions

— Une **Instruction** est un mot-clé compris par le Basic, qui implique l'exécution d'une tâche précise par l'ordinateur. Certaines instructions doivent être suivies d'un argument, certaines non.

Exemples :

Dans PRINT"BONJOUR", PRINT est l'instruction et "BONJOUR" est l'argument. Mais PRINT peut aussi être utilisé seul.

Par contre, CLS (Effacement de l'écran) est une instruction utilisée toujours seule, alors que INPUT implique nécessairement l'emploi d'un ou plusieurs arguments.

— Une **Fonction** est également un mot clé, mais il s'agit plutôt d'une "question" que l'on pose à l'ordinateur : celui-ci doit nous retourner une valeur en réponse. Une fonction s'utilise toujours à la suite d'une instruction, en général PRINT ou LET.

Exemples :

Dans PRINT SIN(PI/2), SIN est la fonction et PI/2 son argument.

A$=HEX$(100) donnera à la variable A$ la valeur hexadécimale de l'argument, ici le nombre 100. (N'oublions pas que A=X est équivalent à LET A=X.)

1.2.2. Variables et constantes

— Les **Opérandes** sont les valeurs, nombres ou séquences de caractères, servant d'argument aux instructions et fonctions. Les opérandes numériques sont de deux sortes, les constantes et les variables:

Les constantes, qui peuvent être soit décimales en notation ordinaire (100, 3.146, etc...) ou scientifique (2.2E3, 4.7E—12), soit hexadécimales si elles sont précédées du symbole # (#20, #B100, etc...).

Les variables, qui contrairement aux constantes peuvent changer de valeur en cours de programme. Leur valeur peut être définie par l'utilisateur, ou représenter le résultat d'un calcul. Une variable est définie par un nom comportant au maximum deux caractères alphanumériques dont le premier doit être une lettre. Deux types de variables sont disponibles:

— Les variables simples, dont la valeur peut être comprise entre 2.9 10^{-39} et 1.7 10^{+38}. A1, N, KM sont des noms de variables simples.

— Les variables entières, qui suivent les mêmes règles que les variables simples avec la différence qu'elles ne peuvent contenir que des nombres entiers compris entre —32768 et +32767, et que leur nom doit être suivi du symbole %; A%, R2% ou AG% sont des noms de variables entières.

Nous avons enfin les opérandes littérales, qui ne sont plus considérées comme des nombres, mais comme des suites de caractères. Il s'agit bien sûr des **chaînes de caractères**, et elles sont aussi divisées en constantes et variables.

Une *constante de chaîne* est une suite quelconque de caractères pouvant aller jusqu'à 255, toujours incluse entre guillemets (pour la distinguer d'un nom de variable).

Exemples:

"PROGRAMME 1", "TOTO", "1234QWERTY" sont des constantes de chaîne.

Les noms des *variables de chaîne* suivent les mêmes règles que ceux des variables numériques, mais ils doivent être suivie du symbole $ (dollar): A$, F1$, etc...

Remarquons que tous les types de variables peuvent être constitués en tableaux. Par exemple, A(3,3) représente un tableau de nombres à deux

dimensions que l'on peut assimiler à un damier 3×3=9 cases, chacune d'entre elles contenant un nombre. Le premier élément ("en haut à gauche") sera A(1,1), et le dernier ("en bas à droite") sera A(3,3). Les nombres entre parenthèses, permettant d'appeler individuellement chaque élément d'un tableau s'appellent des indices; un élément de tableau s'appelle une variable indicée, et il est à noter que les indices peuvent être des constantes (comme dans notre exemple) ou des variables. ORIC accepte les tableaux comportant un nombre quelconque de dimensions. Bien entendu, tous les éléments d'un tableau doivent être de même nature: variables simples, variables entières ou chaînes de caractères. Pour un tableau de chaînes à deux dimensions, il suffit de spécifier le nombre de chaînes qu'il contiendra; la longueur de chacune d'elles est laissée libre.

Enfin, il est intéressant de savoir que des variables de types différents peuvent avoir le même nom: ORIC ne confondra pas A, A% et A$.

1.2.3. Les opérateurs

Les Opérateurs sont de trois types:

Les opérateurs arithmétiques, que tout le monde connaît bien: +, −, * (multiplié par), / (divisé par) et ^ (élévation à une puissance). Ces opérateurs s'utilisent de la manière normale sur les constantes et variables numériques, mais remarquons que "+" peut aussi s'utiliser sur les chaînes. Les chaînes ne sont pas additionnées, mais "concaténées", c'est-à-dire mise l'une à la suite de l'autre.

Exemples:

Si A=123 et B=456, alors A+B=579. Par contre, si A$="123" et B$="456", A$+B$="123456".

Les opérateurs relationnels, >, >=, <, <=, =, <> (supérieur, supérieur ou égal, inférieur, inférieur ou égal, égal, différent). Ces opérateurs peuvent s'utiliser indifféremment sur des nombres ou des chaînes. Dans le cas de deux chaînes, celles-ci sont comparées caractère par caractère, et sont classées par ordre alphabétique comme dans un dictionnaire. Pour ORIC, une condition VRAIE vaut −1, une condition FAUSSE vaut 0.

Exemple:

5>2 est vrai, donc l'expression vaut −1.

"TOTO"<"JOJO" est faux, donc l'expression (5>2)="TOTO">"JOJO" est VRAIE, ainsi que l'expression (5>2)=NOT("TOTO"<"JOJO").

Les opérateurs logiques, AND, OR et NOT (ET, OU et NON) s'utilisent pour combiner des expressions utilisant des opérateurs relationnels.

Exemples :

IF A>2 AND A<5 THEN PRINT A équivaut à : SI A EST COMPRIS ENTRE 2 ET 5 ALORS PRINT A.

1.2.4. Mode direct et mode programme

Dans tous ce qui va suivre, nous nous référerons constamment au mode "direct" et au mode "programme". Rappelons que toute instruction ou séquences d'instructions entrée au clavier et non précédée d'un nombre, est exécutée dès que la touche RETURN est tapée : C'est le mode direct.

Par contre, si une séquence d'instructions est précédée d'un nombre, celui-ci sera considéré comme un numéro de ligne. Le fait de frapper RETURN n'exécutera pas les instructions, mais les mémorisera pour une exécution ultérieure : C'est le mode "différé", ou mode programme ; il faudra taper RUN pour exécuter cette séquence.

Précisons aussi qu'ORIC accepte des lignes de programme de 78 caractères maximum, qu'une même ligne peut contenir plusieurs instructions si celles-ci sont séparées par des : , et que les commandes et noms de variables doivent être tapés en majuscules pour qu'ORIC les accepte.

Enfin, les espaces entre les mots ne sont pas significatifs, contrairemnt à la pratique courante dans d'autres Basic comme le Microsoft. On pourra donc les omettre pour tirer le meilleur parti de la mémoire ou de la longueur de ligne disponible, mais n'oubliez pas que la lisibilité de vos programmes s'en ressentira. (Dans tous les programmes présentés ci-après, des espaces ont été introduits entre les diverses commandes pour des raisons de clarté.)

1.2.5. Pour accroître l'efficacité de vos programmes

Voici quelques renseignements qui intéresseront certainement les fanas de l'efficacité :

— Les variables normales en virgule flottante occupent 7 octets en mémoire, contre 4 pour les variables entières. Par contre, les variables normales sont traitées plus rapidement, plus vite même que les constantes.

— On ne peut utiliser une variable entière pour les boucles. Leur usage est donc surtout conseillé lorsque vous voulez forcer une variable à prendre une valeur entière, en évitant d'utiliser INT. Notons que le résultat est le même que celui fourni par INT, c'est-à-dire arrondi à l'entier inférieur.

— Par contre, un tableau de variables entières n'occupera que les 2/5e de l'espace mémoire exigé pour un tableau identique en virgule flottante : 2 octets par élément contre 5 ; à utiliser de préférence quand l'occupation de la mémoire est critique.

— Le mode de stockage des tableaux de chaînes est identique à celui du Basic Microsoft, à savoir la longueur de chaque élément du tableau n'est pas fixée d'avance, mais peut évoluer en cours de programme. Cette solution s'avère la meilleure en ce qui concerne l'espace occupé (il faut tout de même 3 octets par élément), mais pas pour la rapidité de l'accès.

— En règle générale, et si la vitesse est primordiale, utilisez des variables normales attribuées le plus tôt possible dans le programme et dont le nom ne comporte qu'un seul caractère. Les tableaux sont traités plus rapidement si des variables sont utilisées comme indices.

1.3. Conventions d'écriture

Pour faciliter la compréhension, nous présenterons par la suite les Instructions et Fonctions de la manière suivante :

Syntaxe : Donne la syntaxe complète de la commande.

Application(s) : Donne les détails de son utilisation.

REMARQUES : Dans cette rubrique seront exposées les particularités de la commande, ou les différences avec le Basic Microsoft qui sera pris comme référence en raison de sa très large diffusion.

Exemple : Selon la complexité des cas, un ou plusieurs exemples seront donnés.

— Tous les mots écrits en majuscules représentent les mots-clés du Basic, et sont à entrer tels quels au clavier.

— Les mots écrits en minuscules et compris entre < > sont les données à fournir par le programmeur.

— Les données facultatives sont incluses entre crochets [ ].

— Toutes les ponctuations, hormis les deux précédents, font partie intégrante de la syntaxe et doivent être tapées telles quelles.

— En ce qui concerne les nombres, nous suivrons la syntaxe ORIC : Les nombres décimaux seront écrits normalement, et les nombres hexadécimaux seront précédés de #.

— X, Y, Z représentent des nombres en virgule flottante.

— N, M des nombres entiers.

— A représente une adresse-mémoire (entier compris entre 0 et 65535).

— X$, A$, B$ des chaînes de caractères quelconques.

Exemple : INPUT [<"commentaire";>]<variable 1>[<,variable 2>]

— L'rinstruction INPUT est la commande BASIC.

— Le commentaire est optionnel, sinon il doit être entre guillemets " " et suivi de ; .

— Au moins une variable doit être donnée comme argument.

— On peut si on le désire mentionner plusieurs variables, en les séparant par des virgules.

Ainsi, il serait correct d'écrire :

INPUT"Donnez votre âge et votre mois de naissance";AG,MO$

1.4. Les instructions du Basic

CALL

Syntaxe: CALL <A>,<Nom de variable de chaîne>

Application: Cette instruction appelle une routine en langage machine, préalablement écrite en mémoire et commençant à l'adresse (A) donnée comme argument; la valeur de la variable sera passée à la routine. Il est à noter qu'on ne retournera au Basic que si un RTS (#60, code du 6502 équivalent au "RETURN" du Basic) est rencontré. Des choses amusantes se passeront si vous appelez des routines de la ROM, mais l'ordinateur se bloquera souvent!

Exemple: Tapez CALL 60000: Cela équivaut à un RESET par programme, avec le logo de mise en route en prime.

CSAVE

Syntaxe: CSAVE "<nom du programme>"[,S][,AUTO] ou bien
CSAVE "<nom>",A<adresse début>,E<adresse fin>[,S] [,AUTO]

Application: Dans sa première syntaxe, CSAVE sert à enregisrer sur cassette le programme se trouvant en mémoire. Le nom est à votre discrétion, et peut être une constante ou une variable comportant jusqu'à 17 caractères. L'option AUTO provoquera le démarrage automatique du programme lorsque celui-ci sera rechargé. Si vous utilisez l'option S, la sauvegarde se fera à la vitesse de 300 baud, vitesse très lente offrant une sécurité maximale. Sinon, le transfert se fait au rythme de 2400 baud.

Dans sa deuxième syntaxe, cette instruction puissante permet de sauvegarder une zone-mémoire située n'importe où dans la RAM. On doit donner un nom comme précédemment, et on peut choisir sa vitesse mais l'option AUTO ne devrait être incluse que si la zone contient une routine exécutable en langage-machine. Celle-ci est définie par ses adresses de début (suivant la lettre A) et de fin (suivant la lettre E).

Exemple:

La première application est évidente, mais on mesurera l'importance de la deuxième, qui permet de sauvegarder aussi bien des routines en langage-machine que des jeux de caractères ou des écrans graphiques.

CSAVE"ECRAN",A#A000,E#BF3F sauvegarde l'écran HIRES,
CSAVE"JEU",A#B500,E#BB7F sauvegarde les jeux de caractères.

(N.B. ces valeurs correspondent à la version 48K.)

CLOAD

Syntaxe: CLOAD ["<nom du programme>"][,S] ou bien
CLOAD ["<nom>"][,S],A<adresse début>,E<adresse fin>

Application: Instruction jumelle de CSAVE, celle-ci sert à recharger en mémoire des programmes se trouvant sur cassette. Si un nom n'est pas fourni, ORIC chargera le premier programme rencontré sur la bande. Il est évidemment indispensable de recharger le programme à la même vitesse de transfert que celle utilisée lors de sa sauvegarde: n'oubliez pas le "S"! Sinon, lors du chargement d'une zone-mémoire, les paramètres sont les mêmes que pour CSAVE.

REMARQUE: La manière de sauvegarder et recharger correctement des zones-mémoire est traitée in extenso dans la deuxième partie (§ 2.4.2), consacrée à l'organisation de la mémoire.

Exemple: CLOAD" ",S chargera le premier programme rencontré en vitesse lente.

DEF FN

Syntaxe: DEF FN<nom>(<variable>)=<expression numérique>

Application: Cette instruction permet au programmeur de définir ses propres fonctions, qui seraient éventuellement absentes du Basic. Le nom suit les règles d'appellation des variables; il servira par la suite à identifier la fonction. Pour l'utilisation de la fonction une fois celle-ci définie, voir aussi FN dans le chapitre suivant.

REMARQUES: Pour ceux qui seraient familiers avec d'autres Basic, signalons que les fonctions définies ne peuvent traiter que des nombres, et ne doivent contenir qu'une seule variable; celle-ci est une variable "fantôme", c'est-à-dire que son nom peut être utilisé indépendamment à un autre endroit du programme.

Exemple(s) :

DEF FNTA(X)=SIN(X)/COS(X) calcule la tangente de l'angle X.

DEF FNAS(X)=ATN(X/SQR(–X*X+1)) calcule l'arcsinus de l'angle X.

DEF USR

Syntaxe : DEF USR=<A>

Application : Cette commande permet de définir l'adresse de début (A) d'une routine en virgule flottante en langage machine, avant qu'elle ne soit appelée. Son avantage sur CALL réside dans le fait qu'elle permet aussi bien de transmettre des données à la routine que d'en recevoir de celle-ci. (Voir fonction USR, § 1.5).

DIM

Syntaxe :DIM<tableau 1>(<dim.1>[<,dim.2>]...)[,<tableau 2>(...)]etc.

Application : Pour définir les dimensions des tableaux, avant leur utilisation.

REMARQUES :

— Une fois un tableau dimensionné, toute tentative de le re-dimensionner se soldera par un message d'erreur. Il faut soit remettre tous les éléments du tableau à 0, soit utiliser CLEAR ou RUN qui remettent à 0 toutes les variables. Quel que soit le nombre de ses dimensions, un tableau dont aucune dimension n'a plus de 10 éléments n'a pas besoin d'être déclaré par DIM.

— Rappelons qu'en ce qui concerne les tableaux de chaînes, la chaîne constitue l'élément, et non le caractère comme pour certains autres Basic; la conséquence pratique est que la longueur des chaînes n'a pas besoin d'être définie à l'avance, celle-ci peut donc varier en cours de programme.

Exemples :

DIM A$(15) déclare un tableau à une dimension contenant 15 chaînes.

DIM A(5,5,15),A%(20,20) déclare un tableau numérique en virgule flottante à 3 dimensions, et un tableau d'entiers à deux dimensions.

EDIT

Syntaxe: EDIT<N° de ligne>

Application: Affiche la ligne ayant le N° spécifié.

REMARQUES: Contrairement à l'usage dans d'autres Basic, la commande EDIT sur ORIC est pratiquement équivalente à LIST<N° de ligne>. Seules différences:

— Avec EDIT, le curseur se place au début de la ligne affichée.

— Contrairement à LIST, EDIT vous enverra un message d'erreur si vous demandez une ligne inexistante.

DOKE

Syntaxe: DOKE<A>,<X>

Application: Instruction utile pour charger deux adresses consécutives de la mémoire (A et A+1) avec un nombre X codé sur deux octets. L'adresse A reçoit l'octet de poids faible et A+1 l'octet de poids fort. A et X sont des variables ou constantes comprises entre 0 et 65535, et seront arrondis à l'entier inférieur.

REMARQUE: DOKE A,X est l'équivalent exact de:

```
POKE A,X-(INT(X/256)*256:POKE A+1,INT(X/256)
```

Exemple:

DOKE #26D,48039 place INT(48039/256)=187 à l'adresse #26E, et 48039-(187*256)=167 à l'adresse #26D.

GET

Syntaxe: GET <variable chaîne>

Application: Lorsque cette instruction est rencontrée dans un programme, l'ordinateur attend qu'une touche soit frappée; dès que cela est fait, <variable chaîne> contiendra le caractère correspondant à la touche actionnée et le programme se poursuit.

REMARQUE: Cette commande est utile lorsqu'il n'est pas nécessaire d'exécuter d'autres tâches en attendant une entrée au clavier. Voir aussi la fonction KEY$ au chapitre suivant. L'usage de GET est interdit en mode direct.

Exemple :

```
100 PRINT"POUR CONTINUER, PRESSER UNE TOUCHE"
110 GET X$
.
Suite du programme
.

100 PRINT"VOULEZ-VOUS CONTINUER?(O/N)"
110 GET X$
120 IF X$="N" THEN END
130 IF X$="O" THEN GOTO 150
140 GOTO 100
.
Suite du programme
```

Dans cet exemple, on retournera à la ligne 100 tant qu'une réponse correcte ("O" ou "N") ne sera pas fournie.

GRAB

Syntaxe : GRAB

Application : Récupère la zone-mémoire normalement affectée à l'écran graphique haute résolution pour des programmes ou variables. Cette zone, inutilisée dans les modes TEXT et LORES, n'est pas accessible au Basic à moins d'exécuter un GRAB.

REMARQUE : La zone "libérée" se situe entre les adresses #1800-#3400 sur la version 16K, et #9800-#B400 sur la version 48K. GRAB permet de récupérer 7168 octets pour le Basic.

HIMEM

Syntaxe : HIMEM <adresse>

Application : Définit la taille de la mémoire utile, en fixant l'adresse la plus haute de la zone accessible par le Basic. La zone située au-dessus de HIMEM sera interdite au Basic, donc disponible pour des routines en langage-machine.

REMARQUE : A la mise sous tension, HIMEM est fixé à 40704 (#9FF0) pour la version 48K, c'est-à-dire en plein milieu de l'écran graphique haute-résolution et des choses bizarres se passeront si votre programme utilise beaucoup de chaînes ou tableaux de chaînes. Prenez donc l'habitude de

placer l'instruction HIMEM #97FF au début de vos programmes, ce qui rétablira la valeur correcte. HIMEM est connu aussi sous le nom de RAMTOP ou TOP dans d'autres Basic.

Exemples :

HIMEM 30000
HIMEM #17FF (Valeur correcte sur version 16K)

IF ... THEN ...ELSE

Syntaxe : IF<condition 1>[AND / OR]<condition 2>] .. THEN <instructions> ..

ou IF ..... THEN ..... ELSE <liste d'instructions>

Application : Dans sa première syntaxe, exécutera la série d'instructions situées après THEN si <condition> est VRAIE (=—1), sinon continue l'exécution du programme à la ligne suivante.

Dans la seconde syntaxe, les instructions situées après THEN seront exécutées si <condition> est VRAIE (=—1), et si <condition> est FAUSSE (=0) le programme exécutera les instructions placées après ELSE.

REMARQUES :

— IF<cond.> THEN GOTO <N° de ligne> peut s'écrire indifféremment IF<cond.> THEN <N° de ligne> et aussi IF<cond.> GOTO <N° de ligne>.

— Les IF imbriqués sont autorisés si ELSE n'est pas utilisé.
On peut écrire : IF<cond1>THEN IF<cond2> THEN....
mais *pas* : IF<cond1>THEN...ELSE IF<cond2>..etc..

— ELSE ne peut pas s'utiliser dans tous les cas. ORIC considérera qu'il y a une faute de syntaxe si une variable en virgule flottante est placée juste avant ELSE :

IF<cond.>THEN A=B ELSE.... sera refusé,

par contre,

IF<cond.>THEN A=B:X=1 ELSE....

ainsi que IF<cond.>THEN A=B% ELSE....
seront acceptés.

Exemples:

```
IF A>B AND A<C THEN PRINT"A est compris entre B et C"
IF A/2=INT(A/2) THEN PRINT"A est pair" ELSE PRINT"A est impair"
10 IF X$="OUI" THEN 100 ELSE 200
```

INK

Syntaxe: INK <N> Avec 0<=N<=7

Application: Permet de choisir la couleur des caractères pour la totalité de l'écran, sans perturber l'affichage. Ne modifie pas les couleurs choisies localement par des codes appropriés.

REMARQUE: Pour l'équivalence entre les nombres de 0 à 7 et les couleurs apparaissant à l'écran, voir § 3.4.

Exemple: INK 1 Affiche tous les caractères en rouge.

LIST

Syntaxe: LIST<N° de ligne>
ou LIST<N° de ligne début>–
ou LIST–<N° de ligne fin>
ou LIST<N° de ligne début>–<N° de ligne fin>

Application: Affiche à l'écran les lignes de programme situées entre les limites spécifiées. Si le N° de début est omis le programme est listé depuis le début, sinon lorsque le N° de fin est omis le programme est listé jusqu'à la dernière ligne existante.

REMARQUE: Le fait de fournir des numéros de ligne inexistants, plus petits ou plus grands que ceux du programme ne provoque pas d'erreur.

Exemple:

```
LIST 100-200
```

LORES

Syntaxe: LORES<N> Avec N=0 ou N=1

Application: Sélectionne un des trois modes d'affichage disponibles. Très proche du mode TEXT, avec les différences suivantes:

— L'affichage se fait en vidéo inverse, c'est-à-dire blanc sur fond noir.

— Si N=0, les caractères standards du jeu ASCII seront utilisés.

— Si N=1, l'affichage se fera avec le 2e jeu de caractères (pavés semi-graphiques).

REMARQUE: Si CLS ou CTRL L sont utilisés pour effacer l'écran, on repassera en mode TEXT. Il faut refaire LORES0 ou LORES1 pour effacer l'écran en restant dans le même mode. Voir aussi le § 3.6.1.

ON ... GOTO et ON ... GOSUB

Syntaxe: ON <variable> GOTO <N° de ligne>[,<N° de ligne>,..]..
et ON <variable> GOSUB<N° de ligne>[,<N° de ligne>,..]..

Application: Si <variable>=1, le GOTO ou GOSUB s'effectuera vers la première ligne de la liste, si <variable>=2 vers la deuxième et ainsi de suite.

REMARQUES:

— Si la variable est égale à 0, ou est supérieure au nombre de lignes spécifiées, le programme continuera à la ligne suivante.

— Une erreur sera provoquée si la variable est négative.

— Si la variable provient d'un calcul et comporte une partie décimale, elle sera arrondie à l'entier inférieur. (Voir § 1.6.2 pour la manière de calculer un arrondi exact).

PAPER

Syntaxe: PAPER<N> Avec 0<=N<=7

Application: Instruction complémentaire de INK, permettant de changer la couleur du fond sur tout l'écran sans perturber l'affichage. Comme pour INK, les couleurs choisies localement ne sont pas affectées.

Exemple:

```
10 FOR N=0 TO 7
20 PAPER N:WAIT 50
30 NEXT N
```

Donnera à l'écran successivement chacune des huit couleurs.

PLOT

Syntaxe : PLOT<colonne>,<ligne>,<N> Avec 0>=N>=255
ou PLOT<colonne>,<ligne>,<X$>

Application : Permet d'afficher des données ou des attributs à l'écran, à la position spécifiée par <colonne> et <ligne>.

— Dans la première syntaxe, si N>31, c'est le caractère de code ASCII N qui sera affiché à l'endroit désiré. Si N>159 (31+128), on obtiendra l'affichage en vidéo inverse du caractère de code ASCII N—128. Enfin, si N<32, il sera considéré comme un attribut définissant la couleur, le jeu de caractères, etc. (cf. § 3.4).

— Dans sa deuxième syntaxe, PLOT affichera à l'endroit désiré une chaîne de caractères ou variable-chaîne quelconque.

— Si l'une des deux coordonnées fournies à PLOT est située en dehors des limites de l'écran, une erreur en résultera.

REMARQUES : PLOT n'est pas un équivalent de l'instruction PRINT AT, que l'on trouve sur d'autres machines, dans la mesure ou le curseur n'est pas déplacé après affichage des données. Pour l'adressage direct curseur, voir § 3.5.2.

De plus, PLOT contrairement à PRINT n'accepte qu'une seule donnée à afficher à la fois.

Exemples :

PLOT 10,10,"BONJOUR" est équivalent à
X$="BONJOUR":PLOT 10,10,X$

PLOT 10,10,12:PLOT 11,10,"BONJOUR"
affiche le message au centre de l'écran, en caractères clignotants. (12 est l'attribut "caractères clignotants").

POP

Syntaxe: POP

Application: Supprime la dernière adresse de retour dans la pile des GOSUB. Conséquence pratique: Si un RETURN est rencontré après un POP, on ne retournera pas à l'instruction suivant le dernier GOSUB, mais à l'instruction suivant *l'avant-dernier* GOSUB rencontré si toutefois celui-ci existe; si ce n'est pas le cas, une "RETURN WITHOUT GOSUB ERROR" sera provoquée. A n'utiliser qu'avec précautions!

Exemples:

```
 10 GOSUB 100
 20 PRINT 1
.
.
100 REM 1ER SOUS-PROGRAMME
120 GOSUB 200
130 PRINT 2
140 RETURN
.
.
200 REM 2EME SOUS-PROGRAMME
210 POP:PRINT 3
220 RETURN
```

Normalement, la ligne 220 devrait provoquer le retour à la ligne 130. Mais avec le POP de la ligne 210, c'est à la ligne 20 que le programme reprendra: L'adresse de retour à la ligne 130 a été éliminée.

PULL

Syntaxe: PULL

Application: A exactement la même fonction que POP, mais pour les boucles REPEAT-UNTIL. Si deux boucles REPEAT-UNTIL sont imbriquées, un PULL exécuté une seule fois dans la boucle intérieure provoquera un saut au REPEAT externe. Encore plus traître que POP!

REMARQUE: Un PULL au mauvais endroit vous vaudra un BAD UNTIL ERROR, ou bien engagera le programme dans une boucle infinie. Donc, testez bien vos boucles AVANT d'inclure un PULL.

RELEASE

Syntaxe: RELEASE

Application: Commande complémentaire de GRAB (cf. cette instruction), permet de restituer à l'écran graphique haute-résolution la mémoire que vous lui avez indûment volée pour vos programmes Basic.

REPEAT-UNTIL

Syntaxe: REPEAT

.

lignes de programme

.

.

UNTIL<condition 1>[AND / OR]<condition 2>]...etc

Application: Toutes les lignes de programme situées entre REPEAT et UNTIL sont exécutées en boucle, jusqu'à ce que les conditions situées après UNTIL soient VRAIES (=—1).

REMARQUE: Ces commandes sont l'équivalent ORIC du WHILE-WEND du Basic Microsoft. Avec une différence cependant: Alors que WHILE teste la condition au début, UNTIL teste en fin de boucle. Ce qui implique que même si la condition est fausse avant d'entrer dans la boucle, cette dernière sera exécutée au moins une fois.

Exemple:

```
10 REPEAT
20 X=RND(1)*36+2:Y=RND(1)*26
30 PLOTX,Y,"*"
40 UNTIL X=3
```

Ce programme dessinera des étoiles sur l'écran à des endroits aléatoires, jusqu'à ce qu'une position particulière soit atteinte.

TRON-TROFF

Syntaxe: TRON
TROFF

Application : L'instruction TRON (TRace ON) fournit une aide précieuse à la mise au point des programmes : En effet, les numéros de toutes les lignes de programme exécutées après TRON seront affichés à l'écran en séquence, permettant ainsi de suivre le cheminement du programme et donc de détecter où et pourquoi une erreur se produit. TROFF (TRace OFF) permet d'annuler TRON.

REMARQUE : L'usage de TRON est impossible en mode direct, il faut donc l'incorporer à un endroit propice du programme à tester, et le supprimer ensuite une fois celui-ci au point. Par contre, TROFF peut s'utiliser indifféremment en modes direct ou indirect.

WAIT

Syntaxe : WAIT <N>

Application : Suspend l'exécution du programme pour une durée égale à (N*0.01) secondes. Donc WAIT 1 suspend l'exécution pendant 1/100e de seconde, ou 10 ms.

REMARQUE : Certaines instructions, si elles s'exécutent de façon répétitive dans une boucle, ne peuvent agir correctement que si un WAIT de longueur appropriée est inclus dans la boucle. Il s'agit en particulier de INK, PAPER et des instructions générant un son (cf. 5e partie).

1.5. *Les fonctions du Basic*

FALSE ET TRUE

Syntaxe : FALSE
TRUE

Application : Ce sont deux variables réservées du Basic, au même titre que PI. FALSE (FAUX) vaut 0, et TRUE (VRAI) vaut —1. On les utilise dans les comparaisons avec IF, pour améliorer la lisibilité des programmes entre autres.

Exemple :

```
IF (A>2 AND B=>10)=TRUE THEN.....
```

DEEK

Syntaxe: DEEK(<A>)

Application: Cette fonction nous permet de connaître le contenu de deux adresses-mémoire consécutives (A et A+1), il s'agit bien sûr du complément de l'instruction DOKE. L'adresse A contient l'octet de poids faible, et A+1 l'octet de poids fort. A doit être compris entre 0 et 65534.

REMARQUE: PRINT DEEK(A) est équivalent à:
PRINT PEEK(A)+PEEK(A+1)*256

Exemple:

PRINT DEEK(#A6) vous retournera la valeur de HIMEM

FN

Syntaxe: FN<nom>(<variable>)

Application: Une fonction numérique préalablement définie grâce à DEF FN (cf. cette instruction), s'utilise ensuite comme n'importe quelle autre fonction mathématique standard (SIN ou TAN par exemple) grâce à la fonction FN.

Exemple:

```
10 REM Conversion Degrés → Radians
20 DEF FNDR(X)=(PI*X)/180
.
.
100 A=30
110 PRINT FNDR(A);SIN(FNDR(A))
```

Le résultat affiché sera 0.523598775 0.5, nombres qui sont respectivement la conversion d'un angle de 30° en radians, et le Sinus de 30°.

FRE

Syntaxe: FRE(0)
ou FRE(" ")

Application: Dans sa première syntaxe, cette fonction sert à connaître le nombre d'octets encore inoccupés en mémoire, pendant l'écriture d'un programme par exemple.

Dans sa deuxième syntaxe, la fonction retourne aussi la quantité de mémoire inoccupée, mais en plus force une récupération de l'espace contenant des données inutiles ou supprimées (garbage collection).

REMARQUE: L'ORIC utilise une méthode de rangement des chaînes en mémoire qui a pour caractéristique de ne jamais effacer des données: Les nouvelles données sont ajoutées au précédentes, même si certaines d'entre ces dernières sont supprimées par le Basic en cours de route. Lorsque la totalité de la mémoire est occupée, un "nettoyage" est effectué (appelé "garbage collection"), supprimant toutes les données devenues inutiles. C'est ce nettoyage qu'on peut provoquer artificiellement par l'usage de FRE(" ").

Exemple:

```
10000 IF FRE(0)<200 THEN PRINT"PLUS DE MEMOIRE!"
10010 RETURN
```

Ce sous-programme peut être ajouté à un programme "gourmand" pour vous prévenir quand il n'y a presque plus de mémoire libre.

HEX$

Syntaxe: HEX$(<X>)

Application: Retourne une chaîne représentant la valeur hexadécimale de X.

REMARQUE: X ne peut être ni négatif, ni supérieur à 65535.

Exemple:

PRINT HEX$(65535) nous donne # FFFF

KEY$

Syntaxe: KEY$

Application: Scrute le clavier, mais de façon fugitive: L'exécution du programme continue après que la fonction ait été rencontrée, qu'une touche soit actionnée ou non à ce moment. La fonction doit donc être incluse dans des boucles pour lire efficacement le clavier. C'est la solution idéale pour les jeux, où le clavier doit être constamment surveillé sans pour cela empêcher d'autres actions de s'exécuter simultanément.

REMARQUE: On aura bien sûr reconnu là le INKEY$ des autres Basic, à peine déguisé. Voir aussi l'instruction GET au chapitre précédent.

Exemple:

```
  5 REM DEPLACEMENT D'UN MOBILE SUR L'ECRAN
 10 CLS:X=10:Y=10:A$="<*>"
 20 XM=X:YM=Y
 30 X$=KEY$:A=ASC(X$+" ")
 40 IF A<8 OR A>11 THEN 30
 50 PLOT XM,YM," "
 60 IF A=8 THEN X=X-1
 70 IF A=9 THEN X=X+1
 80 IF A=10 THEN Y=Y+1
 90 IF A=11 THEN Y=Y-1
100 PLOT X,Y,A$:GOTO 20
```

— Les variables XM et YM servent à mémoriser l'ancienne position du mobile, pour l'effacer avant un déplacement.

— A la ligne 30, la variable A prend le code ASCII de la dernière touche pressée. On ajoute systématiquement un espace à X$, car si X$ est une chaîne vide ASC(X$) génère un code d'erreur.

— Ligne 40: Si aucune touche de déplacement n'a été pressée (codes 8 à 11), rien ne se passe.

— Ligne 50: Dans le cas contraire, on efface d'abord le mobile à son ancienne position en y imprimant 3 espaces.

POS

Syntaxe: POS(0)

Application: Retourne la position horizontale du curseur, donc une valeur allant de 0 à 39.

REMARQUE: Cette fonction est inutilisable en mode direct, car après un "RETURN" le curseur retourne à la première colonne de la ligne suivante, et on obtient toujours 0.

RND

Syntaxe: RND(N)

Application: Cette fonction donne des résultats différents selon la valeur de N:

— Si N>=1, retourne un nombre pseudo-aléatoire compris entre 0 et 1 non inclus.

— Si N=0, retourne toujours le nombre précédent.

— Si N<0, le nombre fourni est fonction de la valeur de N, et toujours le même pour des valeurs identiques de N.

Exemple:

```
10 REM DE ELECTRONIQUE
20 PRINT INT(RND(1)*6+1)
30 GET X$:GOTO 20
```

SCRN

Syntaxe: SCRN(<colonne>,<ligne>)

Application: Utilisé dans les modes TEXT et LORES, retourne le code ASCII du caractère situé à la position spécifiée de l'écran.

Exemple: IF SCRN(15,10)=42 THEN EXPLODE

SPC

Syntaxe: SPC(X)

Application: Utilisable uniquement avec PRINT et LPRINT, déplace de X colonnes vers la droite la position d'affichage.

REMARQUE: Dans certains cas, peut remplacer avantageusement la fonction TAB qui marche assez mal sur l'ORIC.

***Exemple*:**

```
PRINT SPC(10)"JE M'APPELLE"SPC(5)"ORIC"
```

TAB

Syntaxe: TAB(X)

Application: Utilisée avec PRINT, cette fonction est théoriquement sensée déplacer le curseur à la Xe colonne d'un ligne.

REMARQUE: En pratique, TAB possède plusieurs caractéristiques bizarres:

— Pour obtenir un positionnement correct, il faut ajouter 12 à la valeur de X.

— On ne peut pas utiliser TAB deux fois sur la même ligne.

Pour des méthodes plus efficaces d'adressage curseur, voir paragraphe 3.5.2.

USR

Syntaxe: USR(X)

Application: La variable en virgule flottante X est passée à une routine en langage machine, dont l'adresse de début a été préalablement définie par DEF USR. La fonction USR retournera ensuite la valeur stockée dans l'accumulateur.

***Exemple*:**

```
10 DEF USR=#400
20 A=22.5
30 PRINT USR(A)
```

MID$

Syntaxe: MID$(A$,N[,M])

Application: Cette fonction très puissante permet d'extraire une partie d'une chaîne de caractères: La chaîne retournée contiendra M caractères de A$, en commençant par le N^e^.

REMARQUES:

— Si M est omis, tous les caractères de A$ à droite du N^e^ seront pris.

— Si N est supérieur à la longueur de A$ ou inférieur à 1, une erreur sera provoquée.

Exemples :

```
10 A$="123456789"
20 PRINT MID$(A$,2,5)
```

Voici pour clore ce chapitre un petit programme qui utilise toutes les fonctions de découpage de chaînes de manière amusante. Vous pourrez facilement l'incorporer comme sous-programme dans vos propres créations :

```
   5 REM CHENILLE LITTERALE
  10 CLS:GOSUB 1000
 100 A$=A1$+A2$+A3$+A4$:N=0:LA=LEN(A$)
 110 WAIT 10:PLOT 1,25,M$
 120 N=N+1:IF N=LA THEN N=1
 130 IF N>=LA-38 THEN M$=RIGHT$(A$,LA-N)+LEFT$(A$,38-(LA-N)):
     GOTO 110
 140 M$=MID$(A$,N,38):GOTO 110

1000 READ A1$,A2$,A3$,A4$:RETURN
1010 DATA "CETTE LONGUE PHRASE EST UNIQUEMENT DESTINEE A TESTER
     LE PROGRAMME, "
1020 DATA "DONT LA SEULE FONCTION EST DE FAIRE DEFILER LADITE
     PHRASE DE DROITE "
1030 DATA "A GAUCHE SUR L'ECRAN, CREANT AINSI UNE ANIMATION DU
     PLUS BEL EFFET: "
1040 DATA"IDEAL POUR LES EXPLICATIONS DANS VOS PROGRAMMES!
     **"
```

Explications :

Le principe du programme est assez simple : On crée une longue chaîne, puis on en extrait une sous-chaîne qui est affichée toujours au même endroit de l'écran. En décalant la sous-chaîne d'un cran vers la droite à chaque fois, on obtient l'effet de défilement désiré.

— 1000 : Lecture de quatre chaînes dans des DATA ; ces quatre chaînes formant une longue phrase sont ensuite ajoutées pour former A$ (ligne 100). Cette technique est indispensable, du fait que l'on ne peut mettre que 78 caractères par ligne de programme ; n'oubliez pas cependant quand vous mettrez vos propres phrases qu'une chaîne ne peut contenir que 255 caractères.

— 140: Chargement de M$ avec un segment de longueur appropriée extrait de A$.

— 130: S'il ne reste plus 38 caractères à prendre à la fin de A$, cette ligne raccorde la fin de A$ avec un bout de longueur appropriée pris à son début: De cette manière, M$ fait toujours 38 caractères de long.

— 120: Lorsque N atteint la fin de A$, on repart au début.

— 110: Introduction d'une petite pause pour améliorer la lisibilité, et impression de M$ à l'endroit désiré.

1.6. *Instructions et fonctions non résidentes*

Malgré la richesse de son "vocabulaire", ORIC n'est cependant pas parfait et certaines commandes disponibles sur les Basic très évolués lui font défaut. Nous avons essayé de sélectionner celles d'entre elles qui nous paraissaient les plus importantes, et dont la simulation par un petit programme reste dans des limites raisonnables.

1.6.1. Instructions

**MID$**(A$,N[,M])=B$

Application:Cette instruction, inverse de la fonction du même nom, permet de remplacer M caractères dans A$, à partir du Nième par les M premiers caractères de B$. Si M est omis, B$ est inséré en entier. Si B$ contient plus de caractères que A$ ne peut en recevoir, les caractères excédentaires seront ignorés.

Équivalent ORIC: L'ORIC possède bien la fonction MID$ permettant de lire une fraction d'une chaîne, mais pas l'instruction permettant de remplacer une partie du contenu d'une chaîne par une autre. L'expression suivante la remplace:

```
A$=LEFT$(A$,N-1)+LEFT$(B$,M)+RIGHT$(A$,LEN(A$)-(N+M-1))
```

PRINT AT<colonne>,<ligne>;<liste de constantes ou variables>

Application: Pour imprimer des données à un endroit précis de l'écran, en y déplaçant le curseur au préalable; ensuite, le curseur reste au même endroit.

Équivalent ORIC: Nous savons que PLOT permet d'imprimer une chose à la fois à une position particulière, mais sans déplacer le curseur; il en va de même pour le programme en langage-machine présenté dans le manuel ORIC. La solution proposée est un véritable adressage curseur, X et Y y représentent les colonne et ligne. Ensuite, il suffit de faire un PRINT pour afficher à l'endroit désiré.

```
POKE 616,Y:?:POKE 617,X
```

Pour plus de précisions sur les variables-système utilisées, voyez le paragraphe 3.5.2.

PRINT USING<chaîne>;<liste de constantes ou variables>

Application: Permet d'afficher des nombres selon un format défini par <chaîne>. L'intérêt principal réside dans le fait que pour les tableaux de nombres, et surtout lorsque ceux-ci comportent des décimales, la lisibilité et la présentation sont grandement améliorées par un format identique (même nombre de décimales, alignement sur la virgule).

Équivalent ORIC: Bien sûr, les vrais PRINT USING autorisent une grande souplesse dans le choix des formats d'impression. Néanmoins, le programme qui suit effectue toutes les opérations d'arrondi exact et de formatage d'un nombre. Celui-ci sera justifié à droite, avec un espace à droite des décimales s'il est positif et le signe moins s'il est négatif, selon la norme de cette instruction. Si un nombre est trop grand pour le format, il sera remplacé par "****".

```
60000 REM PRINT USING
60005 REM F$=FORMAT ("****.**") N=NOMBRE A TRAITER
60010 FOR I=1 TO LEN(F$):IF MID$(F$,I,1)="." THEN
      P1=I-1:P2=LEN(F$)-P1-1:GOTO 60020
60015 NEXT:P1=LEN(F$):P2=0
60020 N=INT((N*10↑P2)+.5)/10↑P2:ST$="**********************"
60030 N$=STR$(N):SN$=LEFT$(N$,1):N$=RIGHT$(N$,LEN(N$)-1):L=LEN(N$)
60040 P=1:REPEAT
60050 P=P+1:UNTIL P>L OR MID$(N$,P,1)="."
60060 NE$=LEFT$(N$,P-1)
60070 IF P<=L THEN ND$=RIGHT$(N$,L-P) ELSE ND$=""
60080 IF LEN(ND$)<=P2 THEN 60110
60090 ND$=ND$+"0"
60100 GOTO 60080
60110 IF LEN(NE$)>=P1 THEN 60130
60120 NE$="  "+NE$:GOTO 60110
60130 IF ND$="" THEN N$=NE$+SN$:GOTO 60150
60140 N$=NE$+"."+ND$+SN$
60150 IF LEN(N$)>P1+P2+2 THEN N$=LEFT$(ST$,P1+P2+1)
60160 RETURN
```

Explication:

— Les variables que vous devez passer au programme sont N, qui est le nombre à traiter, et F$ qui détermine le format; il sert à illustrer les "positions" disponibles pour imprimer N, et peut contenir n'importe quels caractères: L'important est leur nombre avant et après le point décimal.

— Les lignes 60010 et 60015 lisent F$ pour en extraire P1 et P2, représentant respectivement le nombre de chiffres avant et après la virgule.

— La ligne 60020 arrondit N au nombre de décimales souhaitées.

— Ensuite, N est converti en chaîne (N$), puis le signe en est extrait et placé dans SN$.

— Les lignes 60040 et 60130 chargent NE$ et ND$ avec les parties entière et décimale de N$, mises au format.

— Ensuite, le nombre est reconstitué, de manière différente s'il est entier (ligne 60130) et s'il comporte des décimales (lignes 60140).

— Enfin, si N$ est plus long que le format demandé, il est remplacé par une chaîne de même longueur contenant des "*". Si vous ne désirez pas cette option, vous pouvez supprimer la ligne 60150 et la deuxième moitié de 60020.

RENUM<ancien N° début>,<nouveau N° début>, incrément

Application: Renumérote les lignes d'un programme. La renumérotation commence à <ancien N° début>, qui prend la valeur de <nouveau N° début>. Les nouveaux numéros sont séparés de <incrément>.

Équivalent ORIC: Notre programme fera plus de choses que le RENUM standard, puisqu'il vous permet de choisir à la fois la ligne ou commence la renumérotation et la ligne où elle s'arrête. Il a aussi l'avantage d'être court et rapide (environ 7 secondes pour 100 lignes), mais attention: il ne change pas les numéros de ligne suivant les GOTO et GOSUB.

```
60000 REM RENUMEROTATION LIGNES
60010 REM SANS GOTO ET GOSUB
60020 N=#501:CLS
60030 INPUT"Anciens Nos Debut, Fin";DE,FI
60040 INPUT"Nouveau No Début, Increment";DD,IN
60050 A=DEEK(N+2):IF A< DE THEN 60080
60060 IF A=60000 THEN 60090
60070 DOKE N+2,DD:DD=DD+IN:IF A=FI THEN 60090
60080 N=DEEK(N):GOTO 60050
60090 CLS:END
```

Explication:

Ce programme tire parti du fait qu'en mémoire, chaque numéro de ligne est précédé de deux octets contenant l'adresse où commence la ligne suivante. De ce fait, on peut sauter de début de ligne en début de ligne sans être obligé de lire tous les octets intermédiaires, et la vitesse dépend uniquement du nombre de lignes indépendamment de leur longueur.

— N contient les adresses à tester. #501 est la deuxième adresse de la zone programme, début de la première ligne.

— Ligne 60050: A est chargé avec le numéro de la ligne (toujours situé aux adresses N+2 et N+3), et celui-ci est testé pour vérifier s'il n'est pas plus petit que la ligne de début demandée.

— Si le numéro doit être modifié, c'est fait à la ligne 60070. Puis le nouveau numéro est incrémenté et on vérifie que la dernière ligne à modifier n'a pas été atteinte.

— N prend la valeur de la première adresse de la ligne suivante, et on retourne au début de la boucle.

tout cela peut paraître très obscur, surtout si l'on n'est pas familiarisé avec l'organisation mémoire de l'ORIC. Pour mieux comprendre le principe de stockage des programmes en RAM, reportez-vous au paragraphe 2.3.1.

SWAP <variable 1>,<variable 2>

Application: Permet d'intervertir les deux variables données comme argument. Instruction très utile pour les opérations de tri et de classement.

Équivalent ORIC: Une variable intermédiaire (X) est indispensable pour effectuer l'opération, où A et B sont les variables à intervertir:

```
X=A :A=B :B=X
```

1.6.2. Fonctions

Arrondi exact

Nous savons que la fonction INT ne fournit pas un arrondi exact, puisqu'elle arrondit toujours à l'entier inférieur. Voici la formule permettant d'obtenir l'arrondi pour X, D étant le nombre de décimales souhaité:

```
INT((X*10↑D)+.5)/10↑D
```

Si on veut convertir le nombre en entier (D=0), l'expression se réduit à INT(X+.5).

Conversion degrés → radians

Toutes les fonctions trigonométriques de l'ORIC utilisent des angles exprimés en radians. Comme on a souvent besoin de traiter des angles en degrés, voici la formule de conversion:

```
XD=PI*XR/180
```

Où XD est l'angle en degrés et XR l'angle en radians.

INPUT$(N)

Application: Équivalent de l'instruction GET, avec la différence qu'au lieu d'un seul caractère, c'est une chaîne de N caractères qui est saisie au clavier.

Équivalent ORIC:

```
FOR I=1 TO N:GETX$:NEXT I
```

INSTR([N,]A$,B$)

Application: Recherche la chaîne B$ à l'intérieur de A$, à partir de la position N. La valeur retournée est 0 si B$ n'est pas trouvé, ou la position du premier caractère de B$ dans A$.

Équivalent ORIC: Un très court programme remplacera cette fonction indispensable, il pourra être incorporé comme sous-programme dans un ensemble plus important.

```
60000 REM FONCTION INSTR
60005 REM CHERCHE B$ DANS A$ A PARTIR DE I
60007 REM
60010 LA=LEN(A$):LB=LEN(B$):IF I=0 THEN I=1
60020 IF I>LA THEN RETURN
60030 FOR P=I TO LA
60040 IF B$=MID$(A$,P,LB) THEN RETURN
60050 NEXT P:P=0:RETURN
```

A la sortie du sous-programme, P contiendra la position de B$ dans A$, ou 0 si la recherche est négative.

Division modulo

Il s'agit du reste de la division entière entre deux nombres: (10 MOD 4)=2, (23 MOD 5)=3. Qui n'a pas pesté contre les calculatrices (et les ordinateurs!) qui vous sortent toujours une ribambelle de décimales en résultat de la division la plus simple? Voici la solution du problème (expression équivalente à (A MOD B):

```
A-INT(A/B)*B
```

Division entière

Complément naturel de la division modulo, l'équivalent de la division entière (symbole A \ B) est :

INT(A/B)

SPACE$**(N) et **STRING$(N,X$)

Application : STRING$ retourne une chaîne contenant N fois le premier caractère de X$. SPACE$ retourne une chaîne composée de N blancs.

Équivalent ORIC : Il suffit de déclarer en début de programme une chaîne (S$ par ex.), contenant un certain nombre de fois le caractère désiré. Ensuite, il suffira de faire LEFT$(S$,N) pour obtenir le résultat souhaité.

Exemple :

```
 10 S$="----------------------------------------------------
-----"
 20
.
.
100 PRINT LEFT$(S$,10);"RESULTATS";LEFT$(S$,10)
110 PRINT LEFT$(S$,29)
```

Donnera :

```
----------RESULTATS----------
-------------------------------
```

2

L'organisation mémoire et les périphériques

2.1. *Généralités – Rappels*

Pour bien comprendre la manière dont ORIC manipule des informations, il est important de connaître à la fois le type de données qu'il utilise, leur mode de stockage en mémoire et la manière dont celle-ci est structurée. Revoyons rapidement quelques principes de base :

— L'unité d'information la plus élémentaire est le **BIT**, qui ne peut prendre que deux valeurs : 0 et 1. Pour y gagner en vitesse et en efficacité, ceux-ci sont regroupés en "mots" de 8 bits, ou octets.

— Un **Octet** peut prendre 256 (2^8) valeurs différentes, de 0 à 255 ; il constitue l'unité de stockage pour toutes les informations transitant par la mémoire, ainsi que la plus petite unité accessible à l'utilisateur. Certaines données, comme le code d'un caractère par exemple, peuvent se stocker sur un seul octet mais pour des valeurs plus grandes que 255 deux octets ou plus sont nécessaires.

— Chacun des 8 Bits constituant l'octet ne peut valoir que 0 ou 1, mais chacun d'entre eux a un "poids" différent, principe illustré par le schéma suivant:

← OCTET →

Bits →	128	64	32	16	8	4	2	1
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Pour obtenir donc la valeur d'un octet, on multiplie le poids de chacun des bits le constituant par sa valeur (0 ou 1), et on additionne le tout: Si, par exemple, les Bits de "poids" 16, 8 et 1 sont à 1 et tous les autres à 0, l'octet aura la valeur 16+8+1=25. On voit que grâce à cette méthode de codage, un octet peut donc prendre toutes les valeurs entières comprises entre 0 (tous les bits à 0) et 255 (tous les bits à 1).

— Pour savoir où est rangée chaque information et y accéder rapidement, l'ordinateur utilise des **Adresses**: Chaque octet est accessible par son adresse, qui permet de le définir par rapport à ses voisins. Deux octets sont nécessaires si l'on veut stocker une adresse, leurs valeurs étant comprises entre 0 à 65535, donc supérieures à 255. Un des deux octets, dit "de poids fort" contiendra le quotient entier de la division <nombre à coder>/255, et l'autre octet dit "de poids faible" le reste de cette opération. Ces deux octets seront ensuite placés dans deux adresses consécutives, celui de poids faible en premier. Si l'on veut par exemple stocker le nombre 3560 aux adresses 1000 et 1001, il faudra placer la partie entière de la division 3560/256, donc 13 en 1001, et le reste (232) en 1000. L'instruction POKE permet de placer un octet à une adresse, et DOKE évite tout calcul en plaçant un nombre codé sur deux octets dans deux adresses consécutives, conformément à la règle exposée ci-dessus (octet de poids faible dans la première adresse).

2.2. *Le memory-map*

Examinons maintenant la figure 1, qui est un "plan" de la mémoire vive, ou memory-map. On remarque que toutes les données indispensables au

Version 48K — Mode TEXT

#0000:Hexadécimal
(0000) :Décimal

ROM	← **#FFFF (65535)**
	← **#C000 (49152)**
Libre	← **#BFE0 (49120)**
ECRAN	← **#BB80 (48000)**
Jeu de caractères alternatif (graph)	en #26,#27 (621,622)
	← **#B800 (47104)**
Jeu de caractères standard (ASCII)	← **#B400 (46080)**
	← **HIMEM**
Contenu des chaînes	en #A6,#A7 (166,167)
	← en #A0,#A1 (160,161)
TABLEAUX Numériques : valeurs Chaînes : pointeurs	← en #9E,#9F (158,159)
VARIABLES SIMPLES Numériques : valeurs Chaînes : pointeurs	← en #9C,#9D (156,157)
ZONE PROGRAMMES	← en #9A,#9B (154,155)
	← **#500 (1280)**
LIBRE	en #B0,#B1 (176,177)
	← **#400 (1024)**
Adresses Entrées/sort.	← **#300 (768)**
VARIABLES SYSTEME	← **#200 (512)**
PILE	← **#100 (256)**
POINTEURS SYSTEME	← 0

fonctionnement de l'ORIC y sont présentes : ROM contenant l'interpréteur Basic, écran, jeux de caractères, programmes éventuels etc. La mémoire pourrait être comparée à un livre, divisé en chapitres et sous-chapitres, d'inégale importance et traitant de sujets différents. Chacun d'eux n'est susceptible de contenir qu'un type précis d'information. Quant à la zone située en bas de mémoire, entre les adresses 0 et #500, elle représente le "sommaire" du livre (et un peu plus heureusement !) : Tous les octets situés entre ces deux valeurs constituent des "emplacements réservés", où sont stockées toutes les données susceptibles de varier en cours de fonctionnement.

Car en effet, si sur le papier, les différentes parties de la mémoire semblent posséder des limites fixes, en réalité il n'en est rien. Certaines d'entre elles le sont bien sûr, comme celle contenant les variables système ou la ROM ; mais la plupart (celles séparées par des tirets) voient leur taille varier constamment, fluctuant au gré des besoins. Cela paraît évident pour la zone programme par exemple, dont la taille sera bien sûr fonction de l'importance de celui-ci. Pour connaître à tout moment la position de ces adresses variables, ORIC stocke constamment leur valeur dans d'autres adresses, qui elles sont fixes ; ces dernières sont appelées les **pointeurs**.

Sur le memory-map, les adresses fixes sont marquées en **gras**, et les nombres écrits normalement renvoient au pointeur contenant l'adresse considérée.

2.3. *Les différentes zones-mémoire*

Dans ce chapitre sera donnée en détail la fonction de chaque partie spécialisée de la mémoire, la manière dont les différentes données qui la concernent y sont stockées ainsi que la liste des pointeurs ou variables-système s'y trouvant ou s'y référant.

REMARQUE :

— A la mise sous tension, toute la mémoire "utilisateur", c'est-à-dire la zone comprise entre #400 et HIMEM est remplie du caractère "U" (ASCII 85). Ne vous étonnez donc pas de rencontrer souvent 85 si vous faites des PEEK dans cette zone, ou 21845 (85+85*256) si vous utilisez DEEK.

— Lorsqu'une seule adresse est donnée, c'est que son contenu est un nombre inférieur à 255, lu par PEEK et modifié par POKE. Dans le cas de deux adresses consécutives servant elles-mêmes à stocker une adresse, DEEK et DOKE seront bien entendu utilisés.

2.3.1. Les pointeurs-systèmes 0-#FF (0-255)

C'est là que l'on peut trouver tous les pointeurs renseignant sur les limites des zones programme et variables; leur lecture ou modification par le programmeur peut servir simplement à connaître l'espace occupé par un programme, ou aller jusqu'à l'effacement sélectif des variables.

#9A,#9B (154,155)

Fonction: Adresse du premier octet du programme, normalement initialisée à 1281.

Utilisation: Habituellement employée en conjonction avec l'adresse suivante, pour connaître le nombre exact d'octets occupés par un programme, hors variables.

REMARQUE: Cette adresse peut être déplacée, si par exemple vous désirez faire commencer votre programme plus loin. Mais n'oubliez pas de donner la valeur choisie +2 aux pointeurs suivants (#9C, #9E, #A0), sinon des choses étranges se produiront lorsque vous taperez le programme ou entrerez les variables !

#9C,#9D (156,157)

Fonction: Fin de la zone programme, initialisée à 1283 à la mise sous tension.

Utilisation: DEEK(#9C)—DEEK(#9A) retournera l'espace mémoire, en octets, occupé par le programme résident.

REMARQUE: Lors du chargement d'une zone-mémoire par CLOAD A<adresse début>, E<adresse fin>, #9C prend la valeur de <adresse fin>. Concrètement, cela veut dire que si par exemple vous chargez un jeu de caractères, ORIC considérera après le chargement que le programme se termine à l'adresse 47999, où se termine le jeu ! Si ensuite vous tentez de

faire tourner le programme, vous vous retrouverez évidemment avec une "OUT OF MEMORY ERROR". Le moyen de contourner ces difficultés, et d'utiliser les commandes CSAVE et CLOAD pour les zones-mémoire, est expliqué au § 2.4.2.

Comment ORIC mémorise les lignes de programme:

Pour bien illustrer le principe de stockage des lignes, tapons au clavier ce petit programme:

```
10 REM DEMO
20 AB=10:C$="XXX"
30 PRINT AB;C$
```

Maintenant, entrer en mode direct:

```
FOR N=1280 TO DEEK(#9C):PRINT N;PEEK(N);"≠";:NEXT
```

Et les adresses comprises entre 1280 et 1324 s'afficheront à l'écran, chacune suivie de son contenu. Examinons celui-ci:

— 1280: 0, indique le début de la première ligne.

— 1281,1282: Ces deux adresses contiennent 12 et 5, qui donnent 12+5*256=1292. Il s'agit en fait de l'adresse de début de la ligne suivante.

— 1283,1284: Codent le numéro de la ligne, ici 10.

La suite représente le contenu de la ligne proprement dit:

— 1285: 157 est le code de l'instruction REM.

— 1286-1290: Un espace (32), suivi des codes des 4 lettres D, E, M et O.

— 1291: La deuxième ligne commence par 0.

— 1292,1293: C'est là que renvoie le début de la ligne précédente (1281,1282). Cette fois-ci, on trouve les valeurs 31 et 5 donnant 1311, début de la troisième ligne.

— 1294,1295: N° de ligne (20).

— 1296-1309: On trouve ensuite tous les codes des caractères contenus dans la ligne. Remarquons aux adresses 1298 et 1304 que le signe "=" n'est pas représenté par son code ASCII qui est 61, mais par son code d'opérateur (212).

— 1310: Début de la troisième ligne (0).

— 1311-1312 : Les valeurs 43 et 5 donnant 1323 fournissent l'adresse où commencerait la quatrième ligne de programme, si celle-ci existait. De cette manière, si vous ajoutez une ligne son adresse de début sera déjà stockée au début de la précédente.

— 1313-1314 : N° de ligne (30).

— 1315-1321 : Code de l'instruction PRINT (186), suivi des codes de tous les caractères composant la ligne.

— 1322 : Début de la quatrième ligne (0).

— 1323-1324 : Les deux octets sont mis à 0, indiquant la fin du programme.

De tout ce qui précède, on retiendra trois faits importants :

— Chaque instruction, fonction ou opérateur possède son propre code, valeur entière comprise entre 129 et 255 inclus ; ce code est uniquement utilisé dans le stockage des lignes de programme en mémoire.

— Chaque ligne commence par 0, puis deux adresses codant l'adresse de début de la ligne suivante. C'est cette particularité qui a été utilisée dans le programme RENUM (cf. § 1.6), où le programme est renuméroté en sautant directement aux adresses contenant le N° de ligne. D'autre part, et ce quel que soit le programme en mémoire et sa longueur, les adresses 1281 et 1282 (début de la zone) contiendront une valeur nécessairement différente de 0. Si on leur donne cette valeur en tapant DOKE 1281,0 cela équivaut à faire un NEW, puisque l'ordinateur considérera qu'il n'y a plus aucune ligne de programme après cette adresse. La différence avec NEW est que le programme est toujours présent en mémoire, et peut donc être récupéré. (Voir adresse #B0, ci-après).

#9E,#9F (158,159)

Fonction : Fin de la zone des variables simples, initialisée à 1283.

Utilisation : Le début de la zone contenant les variables coïncide avec la fin du programme (#9C), et ne nécessite donc pas de pointeur spécial. Si on élimine la zone des variables en tapant DOKE #9E,DEEK(#9C), on obtiendra le même effet qu'avec CLEAR mais *en conservant les tableaux*.

Mode de stockage des variables simples

Celles-ci sont rangées en commençant par les adresses les plus basses vers les plus hautes.

Les variables en virgule flottante commencent par deux octets contenant le code des deux caractères du nom, puis cinq octets représentant leur valeur. Total occupé : Sept octets.

Pour *les variables entières*, les deux adresses du nom contiennent le code des caractères correspondants plus 128, et la valeur ne nécessite que deux octets puisqu'elle sera comprise entre —32768 et +32767.

Exemple : Soit AB%, dont la valeur est 3500. Elle sera codée sur quatre octets qui seront :

193	Code de A (65) plus 128
194	Code de B (66) plus 128
172	Reste de la division entière 3500/256
13	Quotient entier de 3500/256

Pour les *variables de chaîne*, les choses se compliquent quelque peu, car les adresses les concernant ne constituent que des pointeurs, renvoyant à une autre zone où leur contenu se trouve physiquement :

— Le premier octet est le code du 1er caractère du nom.

— Le deuxième octet est le code du 2e caractère plus 128.

— L'octet suivant contient la longueur de la chaîne, en nombre de caractères (ce qui explique pourquoi les chaînes ne dépassent jamais 255 caractères !).

— Les deux octets suivants codent l'adresse où commencent les données de la chaîne, qui peuvent en fait se trouver n'importe ou dans la zone "contenu des chaînes". Cette adresse combinée avec l'octet précédent donnant la longueur permettent donc de retrouver toujours avec précision ce contenu.

— Enfin, deux octets toujours à 0.

Donc, une chaîne occupera en mémoire un espace égal à sa longueur plus sept octets de pointeurs.

#A0,#A1 (160,161)

Fonction : Contient l'adresse de fin de la zone des tableaux. (Initialisée à 1283 à la mise sous tension).

Utilisation : Cette partie de la mémoire, qui commence là où se termine celle des variables simples, est réservée d'une part aux tableaux numé-

riques et d'autre part aux pointeurs permettant de retrouver les données des tableaux de chaînes.

Stockage des tableaux numériques

Espace occupé	*Données*
2 Octets	Codes des caractères du nom, ou codes plus 128 (tableaux d'entiers).
2	Nombre d'octets occupés par le tableau.
1	Nombre de dimensions du tableau (D), déduit de l'instruction DIM le concernant.
2*D Octets	Nombre d'éléments par dimension (E), également fourni dans DIM.
5*D*E Octets	Valeur de chaque élément du tableau, pour un tableau en virgule flottante, ou bien
2*D*E Octets	Valeur de chaque élément, pour un tableau d'entiers.

Nous voyons dans ce qui précède que les tableaux en virgule flottante sont très "gourmands" en mémoire, puisqu'ils occupent un nombre d'octets égal à 5+2*<nombre dimensions>+5*<nombre d'éléments/dim.> *<nombre dimensions>. Ainsi, un tableau déclaré par DIM A(10,10) ne prendra pas moins de 525 octets, même si tous les éléments ne sont pas utilisés.

Stockage des pointeurs de tableaux de chaînes

Espace occupé	*Données*
2 Octets	Nom du tableau, même que pour les chaînes simples (code du premier car. et code du deuxième car. plus 128).
2 Octets	Espace total pris par les pointeurs.
1 Octet	Nombre de dimensions du tableau (D).
2*D Octets	Nombre de caractères par dimension.
3*D Octets	Pointeurs permettant de retrouver toutes les chaînes du tableau, rangées dans la zone "Contenu des chaînes". — Le 1er octet donne la longueur de la chaîne, — Les deux autres, son adresse de début.

Redimensionner un tableau

Contrairement à d'autres machines, ORIC n'accepte pas de redimensionner (ou effacer) un tableau toute tentative se soldant par un "REDIM'D ARRAY ERROR". Il existe cependant un moyen simple de le faire, si on peut accepter que tous les tableaux (numériques et chaînes) soient effacés en même temps, les variables simples n'étant pas touchées: Il suffit de faire coïncider les pointeurs de début (#9E,#9F) et de fin (#A0,#A1) de la zone "tableaux":

```
DOKE # A0,DEEK(#9E)
```

Vous pouvez ensuite redéclarer les tableaux de votre choix par DIM sans aucun problème.

#A2,#A3 (162,163)

Fonction: Donne l'adresse la plus basse atteinte dans la zone "contenu des chaînes".

Utilisation: DEEK(#A2)—DEEK(#A6) fournit l'espace occupé par les chaînes de caractères. DEEK(#A2)—DEEK(#A0) donnera l'espace demeurant réellement disponible (équivalent à FRE(0).

REMARQUE: Les chaînes sont rangées dans la zone située entre HIMEM (voir ce pointeur) et le haut de la section "tableaux" (codée en #A0,#A1) dans l'ordre de leur arrivée du haut vers le bas, c'est-à-dire en débutant juste au-dessous de HIMEM. C'est l'adresse atteinte par ce remplissage à un moment donné que nous fournissent #A2,#A3. Les zones "programme", "variables" et "pointeurs de chaînes", par contre, s'étendent du bas vers le haut au fur et à mesure des nouvelles entrées. C'est la zone libre entre les deux adresses qui permet la création de nouvelles variables dans un programme; une fois celle-ci remplie, le message "OUT OF MEMORY ERROR" est envoyé.

#A4,#A5 (164,165)

Fonction: Contient l'adresse où commence le stockage des caractères de la dernière chaîne entrée.

Utilisation: La lecture des adresses situées entre DEEK(#A4) et DEEK(#A2) retournera la totalité des caractères composant la dernière chaîne mémorisée, même si ses pointeurs ont été détruits.

HIMEM #A6,#A7 (166,167)

Fonction: Définit à la fois l'adresse la plus haute accessible par le Basic, et le début de la zone de stockage du contenu des chaînes de caractères. (Chaînes simples et tableaux).

Utilisation: Cette adresse est la seule de la mémoire directement accessible par une instruction, c'est dire son importance. En fait, toute la mémoire au-delà de la valeur spécifiée ne sera plus affectée ni par NEW, ni par CLEAR et sera donc tout indiquée pour l'écriture de routines en langage-machine. Seul un RESET remettra tout à zéro et rétablira les limites normales.

REMARQUE: Comme nous l'avons mentionné dans le paragraphe traitant de HIMEM (§ 1.4), l'adresse chargée dans #A6,#A7 à la mise sous tension est erronée. Tapez HIMEM#97FF pour rétablir la valeur correcte, surtout si votre programme utilise de nombreuses chaînes.

#B0,#B1 (176,177)

Fonction: Adresse la plus basse accessible par le Basic, ou LOMEM, initialisée à 1280.

Utilisation: Tout ce qui est situé en dessous de l'adresse contenue dans #B0,#B1 n'est affecté par aucune commande Basic, comme NEW ou CLEAR.

#B8,#B9 (184,185)

Fonction: Adresse de la dernière variable entrée (valeur pour un nombre, et pointeurs pour une chaîne).

Avant de passer à la suite il faut bien comprendre que tous les pointeurs que nous venons d'examiner sont manipulés par l'ordinateur pour toutes les opérations "globales" sur la mémoire. En effet, tout effacement par exemple est obtenu par simple modification des pointeurs définissant la partie à effacer; rien n'empêche le lecteur d'en faire autant en se servant des indications fournies et d'un peu de bon sens.

Exemples :

Émulation de l'instruction NEW

Nous avons vu plus haut que chaque ligne de programme commençait par l'adresse de la ligne suivante, et que la fin du programme est indiquée par la mise à zéro des octets réservés à cet usage. Il suffit donc de mettre à 0 les deux premiers octets du programme, qui devraient normalement contenir l'adresse de la deuxième ligne, pour "effacer" celui-ci :

```
DOKE DEEK(#9A),0        ou plus simplement
DOKE 1281,0
```

Récupérer un programme détruit par NEW

Dans la mesure où cette instruction agit précisément comme décrit dans l'exemple précédent, voici la marche à suivre :

— Placer le début de la zone variables au-delà de la fin estimée de votre programme, par ex. DOKE #9C,1280+5000 si celui-ci fait environ 4 Koctets.

— Lister à l'écran les octets correspondant à la première ligne de programme, en comptant large (un octet par caractère, étant donné qu'une commande n'en occupe qu'un) :

```
FOR N=1281 TO 1281+<Nombre de car.>:? N;PEEK(N);"  ":NEXT
```

— En excluant 1281,1282 qui sont de toutes façons à 0 et 1283,84 qui représentent le numéro de ligne, repérer le premier octet ayant la valeur 0 ; il indique le début de la deuxième ligne du programme.

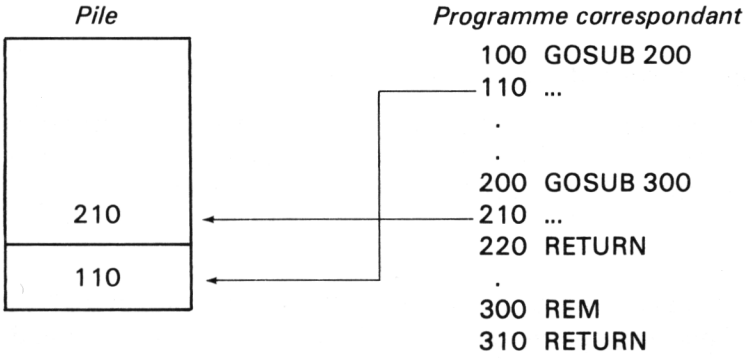
— Charger l'adresse suivant cet octet dans 1281,82 en tapant DOKE 1281,<adresse trouvée>: Votre programme est miraculeusement revenu. A présent, empressez-vous de le sauvegarder car tous les autres pointeurs (ceux des variables notamment) ont également été déplacés par NEW, et il serait malsain de continuer sur de telles bases !

2.3.2. La pile # 100- #1FF (256-511)

Utilisée pour la sauvegarde des résultats intermédiaires de calculs, la pile sert aussi au stockage des adresses de retour pour les GOSUB, REPEAT et

autres FOR-NEXT. En effet, lorsque par exemple un GOSUB est rencontré, l'adresse de début de la ligne suivante ou "adresse de retour" est immédiatement placée dans la pile; le programme saura ainsi où reprendre quand il arrivera à un RETURN, moment où l'adresse désormais inutile est extraite de la pile. En cas de GOSUB multiples ou de boucles FOR-NEXT imbriquées, les adresses de retour sont "empilées" les unes sur les autres, et extraites de telle sorte que la dernière arrivée est toujours la première à ressortir. La taille de la pile n'étant pas infinie, on comprend mieux pourquoi on n'a droit qu'à un nombre donné de boucles ou GOSUB imbriqués.

Exemple:



Lorsque la ligne 100 est exécutée, l'adresse de retour (110) est placée dans la pile. A la ligne 200, c'est au tour de 210 d'y entrer. Ainsi, à l'exécution du RETURN de la ligne 310 on retournera bien en 210, puis ensuite en 110.

2.3.3. Les variables-système # 200-#2FF (512-767)

Véritable tableau de bord et de commandes interne de l'ordinateur, les variables-système sont de loin la zone la plus intéressante à explorer pour le programmeur, par la variété des interventions qu'elles autorisent par rapport au Basic, sur l'affichage et le clavier en particulier.

Avant d'aborder les plus importantes d'entre elles, rappelons que les variables-système mémorisent des données aussi diverses que la taille de

l'écran, le code de la dernière touche enfoncée ou le mode et la position du curseur. On peut toujours se renseigner sur la valeur en cours, mais la plupart du temps on peut également modifier celle-ci avec l'instruction POKE et c'est bien entendu cette dernière caractéristique la plus intéressante.

Les diverses variables seront regroupées en celles concernant l'écran et celles concernant le clavier plutôt que par ordre croissant, autorisant ainsi une consultation aisée.

Le clavier

#208 (520)

Fonction: Contient le code de la touche frappée au clavier.

Utilisation: La lecture de cette variable par PEEK fournit une alternative intéressante à la fonction KEY$, dans tous les cas où la détection rapide de l'appui sur une touche est primordiale (jeux en temps réel).

#209 (521)

Fonction: Prend la valeur 162 si la touche CTRL est appuyée, et 167 s'il s'agit de SHIFT. Pour toute autre touche du clavier, a la valeur 56. (Bits 3, 4 et 5 à 1 et tous les autres à 0).

Utilisation: La lecture de cette adresse est le seul moyen de détecter l'usage des touches SHIFT et CTRL, sans être tenu de tester le caractère généré.

#20C (524)

Fonction: Prend la valeur 127 en mode minuscules, et 255 en mode "blocage majuscules".

Utilisation: Très utile pour forcer le clavier dans le mode désiré sans tenir compte de son état présent, contrairement à CTRL T qui agit en bascule.

#20E (526)

Fonction: Décompte le temps d'appui sur une touche, et déclenche l'auto-répétition le cas échéant.

Utilisation Au repos (aucune touche actionnée), l'octet a la valeur 32 (Bit 5 à 1). Si une touche est pressée:

— L'octet est décrémenté d'une unité à chaque scanning clavier;

— Lorsque la valeur 1 est atteinte, l'autorepeat est déclenché. Tant que celui-ci dure, seuls le Bits 0, 1 et 3 pourront être à 1.

Concrètement, cela signifie qu'il est aisé de détecter l'appui sur une touche (octet<32), ainsi que la mise en route de l'autorepeat (octet<5). On peut aussi forcer la répétition dès qu'une touche est actionnée:

Exemple:

```
10 X$=KEY$
20 POKE 526,1:PRINT X$;
30 GOTO 10
```

— Pour toute autre valeur que 1 donnée comme argument à POKE 526, la répétition est bloquée.

— On peut ralentir la vitesse à volonté en insérant une instruction WAIT N avant GOTO 10, avec 1<N<15.

#306,#307 (774,775)

Fonction: Détermine la vitesse d'autorépétition du clavier ainsi que celle de clignotement du curseur.

Utilisation: Il faut bien comprendre que ces deux adresses ne stockent pas des données fugitives comme #20E, mais une valeur qui est prise en compte tant qu'elle n'est pas modifiée par le programmeur ou réinitialisée par RESET. La valeur normale est de 10000 sur les deux adresses, mais #306 étant l'octet de poids faible il pourra être ignoré en pratique, pour n'agir que sur #307. Sa valeur normale est alors de 39, valeur qu'il faudra diminuer (jusqu'à 10 environ) pour accélérer, et augmenter pour ralentir la vitesse de répétition.

L'Écran

Note: Les applications de toutes les adresses décrites dans cette rubrique sont développées en détail dans la troisième partie concernant l'écran.

#12,#13 (18,19)

Fonction: Contient l'adresse de début de la ligne courante (ligne où se trouve le curseur), dans la partie de la mémoire concernant le contenu de l'écran.

Utilisation: Du fait qu'il n'existe aucune fonction renseignant sur la position du curseur à l'écran, DEEK(#12) nous fournira la ligne où celui-ci se trouve en mémoire, comme PEEK(616) retourne la ligne du curseur à l'écran (cf. cette adresse, ci-après).

#268 (616)

Fonction: Retourne la ligne où se trouve le curseur à l'écran, de 1 à 27.

Utilisation: Cette adresse très importante autorise l'adressage vertical curseur, et un adressage complet en conjonction avec #269 (voir ci-dessous). Le placement du curseur n'est effectif qu'après un PRINT, il faut donc donner le numéro de ligne désiré —1 car PRINT provoque un saut de ligne; Pour plus de détails sur tous les modes possibles de déplacement du curseur voir paragraphe 3.5.2.

Exemples: Pour bien comprendre le fonctionnement de POKE 616, essayez les deux lignes suivantes en mode direct, après avoir mis le curseur vers le milieu de l'écran à l'aide des flèches:

```
POKE 616,0:?"A":?"B" et
POKE 616,0:?:?"A";"B"
```

#269 (617)

Fonction: Contient le numéro de la colonne où se trouve le curseur, de 0 à 39.

Utilisation: Comme l'adresse précédente, place aussi le curseur à l'écran mais horizontalement. Cette fois-ci, le positionnement est immédiatement pris en compte et POKE 617,<colonne> procure une alternative efficace à la fonction TAB, qui comme nous l'avons vu est quelque peu défaillante. Évidemment, l'essai en mode direct ne donnera aucun résultat puisque le curseur passe à la ligne suivante, colonne 0 une fois la commande exécutée. Pour voir son effet, il faut essayer une ligne du type 617,20:?"XXXX".

#26A (618)

Fonction: Contient toutes les données relatives au mode curseur, correspondant à certains codes CTRL du clavier. Voici celles qui ont pu être déterminées:

Bit	*Valeur*	*CTRL*	*Fonction*
1	0 1	Q	Curseur invisible Curseur visible
2	0 1	S	Blocage de l'affichage Déblocage de l'affichage
4		P	??
8	0 1	F	Rappel sonore au clavier Clavier muet
16			??
32	0 1	]	Affichage en 38 colonnes Affichage en 40 colonnes
64	0 1	D	Affichage en simple hauteur Affichage en double hauteur
128			??

Utilisation: Tous les codes CTRL ayant des effets équivalents à ce qu'on peut obtenir avec POKE 618 agissent comme des bascules, il est donc souvent intéressant de pouvoir "forcer" tel ou tel mode indépendamment du mode où l'on se trouve. A la mise sous tension #26A a la valeur 3, qui correspond bien à un curseur visible et clavier sonore. Voir aussi codage d'un octet, § 2.1.

REMARQUE: Le codage représenté dans le tableau est un codage "bit à bit", chaque bit de l'octet présent à l'adresse 618 étant suivant sa valeur (0 ou 1) associé à un mode curseur. "1" est le bit de poids faible, et 128 celui de poids fort.

Exemple: POKE 618,74 démarre la double hauteur (64), rend le clavier muet (8), et garde l'affichage actif (2): 64+8+2=74. Le curseur disparaît (Bit de poids 1 à 0, d'une manière générale toutes les valeurs impaires sont "curseur visible").

Les codes CTRL sont traités in extenso au paragraphe 3.2.1.

#26B et #26C (619-620)

Fonction: Contiennent respectivement l'attribut du fond et de l'affichage, pour tout l'écran. 619 contient l'octet placé dans toute la première colonne de l'écran, et 620 celui de la deuxième. Leur valeurs normales sont respectivement 23 et 0.

REMARQUE: L'effet de l'attribut donné à 619 ou 620 n'est visible qu'après un effacement de l'écran, avec CTRL L ou CLS.

Utilisation: Lorsqu'il s'agit de couleurs, PAPER et INK sont bien plus commodes, mais en pokant dans ces adresses on peut aussi bien faire clignoter l'écran, ou passer en caractères géants. (Voir aussi § 3.6.2.).

Exemple:

```
POKE 618,67:POKE 619,10:INK 7:CLS
```

#26D,#26E (621,622)

Fonction: Spécifie la première ligne de l'écran accessible au curseur. La valeur normale est de 48000, laissant la ligne 0 inaccessible.

Utilisation: Grâce à ces deux adresses ainsi qu'à la suivante (voir ci-dessous), l'utilisateur peut définir quelle taille aura l'écran, et même la changer instantanément. La ligne s'obtient par 47960+<ligne>*40.

Exemple: Pour "récupérer" la ligne 0, taper DOKE # 26D,47960.

#26F (623)

Fonction: Spécifie la dernière ligne de l'écran accessible au curseur, la valeur normale étant 27.

Utilisation: Adresse complémentaire des deux précédentes, elle permet d'interdire à l'affichage un nombre de lignes quelconque en bas d'écran. Le

scrolling intervient sur la nouvelle ligne fixée, comme s'il s'agissait réellement de la dernière.

Exemple: Éteindre et rallumer l'ORIC, puis taper:

```
POKE 623,10:FOR N=1 TO 10:?:NEXT
```

#2C0 (704)

Fonction: Prend la valeur 2 en mode TEXT et 3 en HIRES.

Utilisation: C'est cette variable qui détermine le rejet des commandes graphiques en mode texte, le fait de lui donner la valeur 3 permet de dessiner à partir de ce mode sans voir l'écran haute-résolution.

2.3.4. Les autres zones de la mémoire

#300-#3FF (768-1023)

Dans cette section sont situés tous les pointeurs utilisés par le système d'exploitation de cassettes (SEC) et par l'interface parallèle de l'imprimante dont les fonctions sont passées en revue au paragraphe 2.4.1., ainsi que pour les communications internes dans le détail desquelles nous n'entrerons pas.

#400-#4FF (1024-1279)

Cette zone est laissée libre, permettant à l'utilisateur de stocker ses propres routines en langage-machine, ou toutes sortes de données que l'on veut rendre inaccessibles à NEW, RUN ou CLEAR. Ces adresses sont fréquemment employées dans les programmes qui suivent.

La mémoire utilisateur

#500-#B3FF (1280-46080)

Il s'agit d'une section qui totalise quelque 44 Koctets exploitables regroupant programmes et données, et dont seules les deux limites extrêmes sont fixes, les autres étant gérées par les pointeurs qui ont été traités au paragraphe 2.1.1.

Les jeux de caractères et l'écran

#B400-#BFE0 (46080-49120)

La zone située entre #B400 et #BB7F regroupe le dessin de tous les caractères utilisés par ORIC, chaque caractère étant codé sur 8 octets (8×8=64 points/car.). Ceux-ci, ainsi que la manière de les modifier, sont vus au paragraphe 3.7. Il en va de même pour la mémoire de l'écran texte, où chaque octet représente une position d'affichage et située entre #BB80 et #BFE0, examinée au paragraphe 3.1.

La ROM (mémoire morte)

Suivent une trentaine d'octets également disponibles pour le programmeur situés entre #BFE0 et #BFFF, et nous arrivons à la ROM. Ou plutôt, serait-il plus correct de dire "l'image de la ROM en RAM", car en effet cette dernière, tout comme les jeux de caractères et toutes les variables situées entre 0 et #500 est recopiée en RAM lors de la mise sous tension, laissant ainsi le champ libre pour d'éventuelles modifications. Ces dernières (sur les jeux de caractères notamment) demeureront effectives jusqu'à l'extinction de la machine ou un RESET.

Simulation de RESET par programme

Deux routines de la ROM, appelées par CALL<adresse> permettent d'obtenir ce résultat:

#F84A (63562): Effectue un RESET total, avec remise de toute la mémoire à l'état initial et perte du programme et des variables. Le logo Tangerine est affiché en fin d'opération, comme à la mise sous tension.

#F882 (63618): Équivalent de RESET, remet la mémoire à l'état initial sans perte des variables et du programme et sans affichage du logo.

2.4. La gestion des périphériques

Les quatre périphériques actuellement gérés par l'ORIC sont le clavier, l'écran, l'interface parallèle imprimante et le magnétophone K7. Les diffé-

rentes variables et instructions concernant le clavier ayant été vues plus haut et la troisième partie étant entièrement consacrée à l'écran, nous traiterons ici de l'imprimante et du K7.

2.4.1. L'imprimante

L'interface disponible sur l'ORIC est du type Centronics parallèle, autorisant la connexion directe à un large éventail d'imprimantes sans adaptation particulière. Deux instructions du Basic sont fournies pour son exploitation.

LPRINT, qui agit de la même manière que PRINT en permettant l'envoi de tous les caractères ASCII vers l'imprimante, y compris les caractères de contrôle de cette dernière grâce à LPRINT CHR$(N), N étant compris entre 0 et 31.

LLIST, qui listera le programme sur l'imprimante et dont la syntaxe est identique à celle de LIST.

Remarquons que sur une certaine quantité de données imprimées, on voit souvent apparaître un caractère parasite qui a le code 127. Sur certaines imprimantes cela correspond à DEL, et provoque la suppression d'un caractère alors que sur d'autres il s'agit du caractère ¨. Ceci est provoqué par une interférence entre la lecture très rapide (scanning) du clavier, et l'interface parallèle. Il y a heureusement un moyen d'éviter ce problème, aussi bien lors de l'impression de données avec LPRINT que lors de l'impression de programmes avec LLIST: La lecture du clavier est désactivée par CALL#ED01, et réactivée avec CALL#E804.

Attention : Une fois la première routine appelée le clavier reste évidemment bloqué, il faut donc entrer un ligne complète du type CALL#ED01:LLIST:CALL#E804 en mode direct. En tout état de cause, un RESET vous rendra également la main.

Soulignons enfin que toutes les fonctions utilisables avec PRINT le sont aussi avec LPRINT, y compris celles concernant le placement horizontal du curseur telles que TAB ou SPC.

Pour des applications plus complexes utilisant l'imprimante, voir aussi paragraphe 3.4.1 (hardcopy texte) et paragraphe 4.4 (hardcopy graphique).

Transfert de zones-mémoire sur l'imprimante

Le programme qui suit listera sur l'imprimante toutes les adresses situées entre des limites fixées par vous, suivies de leur contenu, et avec la meilleure présentation possible (Dump mémoire).

```
 10 REM DUMPS MEM
 12 REM
 20 CLS:?:?"DUMPS MEMOIRE SUR IMPRIMANTE":?:?
 30 INPUT"ADRESSE DEBUT,ADRESSE FIN";AD,AF
 40 A=10-LEN(STR$(AF))
 50 ?:INPUT"DESCRIPTIF DE LA ZONE";M$
 60 LPRINT"ORIC1 -- DUMP ZONE"M$" "HEX$(AD)"-"HEX$(AF)
 70 LPRINT:LPRINT:LPRINT CHR$(27)"N"CHR$(3):CALL  ED01
100 FOR N=AD TO AF STEP 6
110 A$=STR$(PEEK(N)):LPRINT N"="A$SPC(A-LEN(A$));
120 A$=STR$(PEEK(N+1)):LPRINT N+1"="A$SPC(A-LEN(A$));
130 A$=STR$(PEEK(N+2)):LPRINT N+2"="A$SPC(A-LEN(A$));
140 A$=STR$(PEEK(N+3)):LPRINT N+3"="A$SPC(A-LEN(A$));
150 A$=STR$(PEEK(N+4)):LPRINT N+4"="A$SPC(A-LEN(A$));
160 A$=STR$(PEEK(N+5)):LRPINT N+5"="A$SPC(A-LEN(A$));
200 NEXT:CALL#E804
```

Explications:

— AD est l'adresse de début, AF l'adresse de fin; A$ est une chaîne formattée (4 caractères), contenant l'octet à imprimer: Tous les nombres sont ainsi alignés en colonnes.

— La ligne 70 est spécifique EPSON: ESC"N"+CHR$(X) saute X espaces à la fin de chaque page, évitant ainsi d'écrire sur les pointillés. Remplacez cette séquence par celle remplissant la même fonction sur votre imprimante, ou mettez simplement trois LPRINT.

2.4.2. Le magnétophone à cassettes

Celui-ci est géré par deux instructions Basic, CSAVE et CLOAD, dont les applications sont beaucoup plus nombreuses qu'il ne peut y paraître à première vue. Leur utilisation normale pour la sauvegarde et lecture de programmes étant traitée au paragraphe 1.4, nous nous intéresserons plus particulièrement à la sauvegarde de zones-mémoire et de tableaux de variables.

CSAVE

Syntaxe: CSAVE"<Nom>",A<adresse début>,E<adresse fin>[,S] [,AUTO]

Fonction: Sauvegarde toute la zone-mémoire comprise entre <adresse début> et <adresse fin>, ces deux dernières y compris.

Utilisation: Celle-ci n'appelle pas de commentaires particuliers, en dehors du fait que l'option AUTO ne devrait être utilisée que si la zone sauvegardée contient une routine exécutable en langage-machine. S demande la sauvegarde lente (300 baud), l'omission impliquant une sauvegarde rapide (2400 baud). Précisons aussi que <adresse début> et <adresse fin> peuvent être des constantes décimales ou hexadécimales des variables ou des expressions numériques.

CLOAD

Syntaxe: CLOAD ["<Nom">],A<adresse début>,E<adresse fin>[,S]

Fonction: Recharge la zone-mémoire portant le nom spécifié, ou la première rencontrée si celui-ci est omis.

Utilisation: Elle appelle nettement plus de commentaires que CSAVE, dans l'ordre:

- Les valeurs données comme arguments de A et E importent peu, la zone considérée étant de toute manière replacée à l'endroit d'où elle a été sauvegardée par CSAVE. L'intérêt réside dans le fait qu'il n'est absolument pas nécessaire de savoir d'où provient une zone pour la recharger à l'emplacement correct, le revers de la médaille étant l'impossibilité de reloger cette dernière à un endroit différent de sa première implantation.
- Il n'y a aucun moyen de rester à l'intérieur d'un programme après CLOAD, celui-ci provoquant automatiquement le retour en mode direct après son exécution si AUTO est omis dans la sauvegarde, omission indispensable dans le cas d'une zone-mémoire non exécutable (écran ou jeu de caractères par ex.).
- CLOAD modifie les pointeurs de fin de programme et de variables. Cela n'a que peu d'importance si un programme est chargé ensuite, mais peut être désastreux si une zone est chargée à partir d'un programme résident. Dans ce cas:

— Les variables simples et les tableaux de chaînes sont perdus.

— Le pointeur de fin de programme prend la valeur de <adresse fin>, donnée dans le CSAVE qui a sauvegardé la zone. Donc, si vous chargez un écran ou un jeu de caractères, ORIC considérera que le programme se termine à l'adresse 48000 ou 49000, et toute tentative d'exécuter un RUN provoquera une impitoyable OUT OF MEMORY ERROR.

Il existe deux manières de contourner ce problème :

1)

— Prévoir une ligne 1 dans le programme.

— Repérer la valeur contenue dans #9C,#9D par ?DEEK(#9C) avant de charger la zone désirée.

— Charger la zone, par exemple avec CLOAD"<Nom>",A0,E0 (les arguments de A et E n'ont pas d'importance).

— Redonner la valeur correcte à #9C,#9D, par DOKE#9C,<valeur lue>.

— Relancer le programme en tapant GOTO 1.

2)

Ce procédé est plus élégant car automatique, nous l'avons d'ailleurs utilisé dans tous les programmes par la suite :

— Insérer dans le programme la ligne suivante :

```
1 IF DEEK(#9C)>40000 THEN DOKE#9C,DEEK(#400) ELSE DOKE#400,
  DEEK(#9C)
```

A la première exécution du programme, l'adresse stockée dans #9C,#9D est forcément correcte, donc inférieure à 40000. Dans ce cas, on mémorise cette adresse de fin de notre programme dans deux adresses libres (#400,#401). Ensuite, après le CLOAD, #9C aura une valeur erronée (>40000) et il ne reste plus qu'à y replacer la valeur correcte en lisant la valeur mise précédemment dans #400,#401.

— Relancer le programme en mode direct avec GOTO 1.

Il n'est pas important que la ligne ci-dessus ait le numéro 1, mais elle doit en tout état de cause être située avant toute attribution de variables par le programme, qui provoqueront une OUT OF MEMORY ERROR si la valeur correcte de #9C,#9D n'a pas été restituée.

Exemples : Il est parfaitement possible de sauvegarder et de recharger des zones en mode direct.

Pour sauvegarder un écran-texte:

CSAVE"<Nom>",A#BB80,E#BFE0 et relecture par:
CLOAD"<Nom>,A0,E0

Sauvegarde d'un jeu de caractères:

CSAVE"<Nom>",A#B400,E#BB7F relu par:
CLOAD"<Nom>",A0,E0

Voir aussi les diverses applications de ces méthodes dans les programme donnés plus loin, CARGEN et MULTIGRAF.

Pointeurs associés au K7

Leur connaissance n'est pas indispensable, mais peut se révéler utile pour ceux qui programment en langage-machine:

#35-#46 (53-70)

Nom du programme (17 caractères max.). La dernière adresse du nom doit toujours être 0.

#5F,#60 (95,96)

Adresse de début de la zone à sauvegarder.

#61,#62 (97,98)

Adresse de fin.

#63 (99)

doit être à 1 si AUTO, à 0 dans le cas contraire.

#64 (100)

A 0 s'il s'agit d'un programme Basic, et à 1 dans le cas d'une zone (option A et E).

#67 (103)

A 0 si vitesse rapide, à 1 si vitesse lente (option S).

Voici les routines en ROM effectuant les opérations d'entrée-sortie K7 :

Sauvegarde :

CALL#E6CA Initialisation du VIA et blocage du clavier
CALL#E57B Effectue la sauvegarde
CALL#E804 Initialise le clavier et rend la main

Relecture :

CALL#E6CA """"
CALL#E4A8 Chargement du fichier
CALL#E804 """"

REMARQUE : La routine de chargement modifie les pointeurs du Basic de la même manière que CLOAD, les mêmes précautions devront donc être prises si elle est utilisée par l'intermédiaire d'un programme Basic.

La sauvegarde et le chargement de variables sur K7

Les instructions effectuant ces opérations font cruellement défaut à l'ORIC, mais heureusement une routine (assez complexe) en langage-machine nous sortira de ce mauvais pas :

— La routine fonctionne pour les tableaux de chaînes quels que soient leur taille et le nombre de leurs dimensions.

— Elle sera implantée à la place du deuxième jeu de caractères entre #B800 et #B9A7 inclus, il sera donc impossible d'utiliser ce dernier ou la haute résolution.

— Nous avons fait commencer le sous-programme Basic effectuant l'implantation à la ligne 50000, il sera donc appelé en début du programme principal par GOSUB 50000, mais vous pouvez lui donner les numéros de ligne de votre choix.

— Une fois la routine implantée, on l'utilise avec :

CALL#B9A2, <Nom du tableau> pour sauvegarder.

CALL#B9A5, <Nom du tableau> pour recharger.

Où <Nom du tableau> est le nom du tableau de chaîne a sauvegarder ou celui qui reçevra les données à la relecture, et répondant à la syntaxe ORIC (A$, TA$, etc...).

POKE #67,1 sélectionnera éventuellement la vitesse lente (S) pour la sauvegarde ou la relecture.

— Bien entendu, pour que le chargement s'effectue correctement le tableau doit être dimensionné au préalable par DIM avec exactement les mêmes paramètres que lors de sa sauvegarde.

```
50000 REM **SAVE-LOAD TABLEAUX**
50002 REM
50010 DE=#B800
50020 READ VA:REPEAT
50030 POKE DE,VA:DE=DE+1
50040 READ VA:UNTIL VA=33
50050 DOKE#B9A2,#0A4C:DOKE#B9A4,#4CB8:DOKE#B9A6,#B858
50060 RETURN
50100 DATA 85,85,85,85,35,57,68,54,56,85,32,11,185,8,32,214,184,32,186,230,169
50110 DATA 37,32,198,229,165,51,32,198,229,165,52,32,198,229,32,238,184,32,167
50120 DATA 229,36,40,16,3,32,53,184,32,4,232,40,96,160,0,177,1,240,23,170,160,2
50130 DATA 177,1,153,208,0,136,208,248,232,202,240,8,177,209,32,198,229,200,
      208
50140 DATA 245,32,195,184,144,222,96,32,149,213,32,11,185,8,32,214,184,32,150
50150 DATA 230,32,48,230,201,37,208,249,32,48,230,133,51,32,48,230,133,52,160
50160 DATA     2,177,206,197,51,200,177,206,229,52,176,6,32,4,232,76,131,196,32,
      238
50170 DATA 184,32,235,228,36,40,16,3,32,155,184,32,4,232,40,96,160,0,177,1,240
50180 DATA 28,32,240,212,170,232,160,0,202,240,8,32,48,230,145,209,200,208,245
50190 DATA 160,2,185,208,0,145,1,136,208,248,32,195,184,144,217,96,24,169,3,
      101
50200 DATA 1,133,1,168,144,2,230,2,165,2,196,97,229,98,96,32,202,230,32,24,185
50210 DATA 160,3,177,206,170,136,177,206,233,1,176,1,202,133,51,134,52,96,24,
      165
50220 DATA 206,101,51,133,97,165,207,101,52,133,98,160,4,177,206,32,246,209,
      133
50230 DATA 95,132,96,133,1,132,2,96,32,232,0,201,44,240,3,76,228,207,76,226,0
50240 DATA 162,0,32,232,0,134,39,133,180,32,232,0,32,134,209,176,6,32,4,232,76
50250 DATA 228,207,162,0,134,40,134,41,32,226,0,144,5,32,134,209,144,11,170,32
50260 DATA 226,0,144,251,32,134,209,176,246,201,36,208,6,169,255,133,40,208,12
50270 DATA 201,37,208,15,169,128,133,41,5,180,133,180,138,9,128,170,32,226,0
50280 DATA 134,181,166,158,165,159,134,206,133,207,197,161,208,4,228,160,240,31
50290 DATA 160,0,177,206,200,197,180,208,6,165,181,209,206,240,14,200,177,206
50300 DATA 24,101,206,170,200,177,206,101,207,144,215,56,96,32,4,232,162,42,76
50310 DATA 133,196,85,33
```

Comme il pourrait être dangereux pour votre équilibre mental d'avoir à taper ce programme deux fois de suite, suivez ces quelques conseils:

— Dès que vous avez fini de le taper sauvegardez immédiatement le programme, de préférence en double exemplaire sur deux cassettes différentes *avant de le tester*. En effet, si vous avez commis la moindre faute de frappe et qu'un incident se produisait, celui-ci serait irrémédiablement perdu. Il vous sera facile ensuite de le recharger et de le corriger tout à loisir.

— Il est préférable, avant de le taper ou de le charger, de protéger la zone d'implantation de la routine par HIMEM#97FF.

— Avant de commencer l'écriture d'un autre programme pouvant nécessiter la sauvegarder de variables, chargez d'abord celui-ci, même si vous n'êtes pas absolument sûr d'en avoir besoin. Il est beaucoup plus facile de l'effacer s'il ne sert à rien que de le retaper s'il s'avère indispensable !

— Si les numéros de ligne ne conviennent pas, ils seront très facilement changés grâce au programme RENUM donné au paragraphe 1.6.1. Comme le sous-programme ci-dessus ne comprend aucun GOTO ou GOSUB, il n'y aura aucun souci à se faire à ce sujet.

Enfin, cette routine est utilisée "sur le terrain" dans le programme GENETEX, paragraphe 3.8.2.

3

L'écran et les jeux de caractères

Cette section est consacrée à l'écran en modes TEXT et LORES, ainsi qu'à tous les codes, commandes et fonctions correspondants ; l'écran haute résolution graphique est traité en quatrième partie.

3.1. La représentation de l'écran en mémoire

L'écran, tel que nous le voyons sur un téléviseur, est constitué comme un rectangle divisé en 28 lignes de 40 colonnes, donc un total de 28*40=1120 cases. Un caractère ne pourra s'imprimer que dans une de ces "positions d'affichage", qui sera définie par son numéro de ligne et colonne.

Or, il faut savoir que l'écran est entièrement représenté en mémoire, chaque case de celui-ci occupant un octet. Si on place par POKE le code ASCII d'un caractère dans une des adresses de cette zone-mémoire, le

caractère s'affichera à la position correspondante sur l'écran. Inversement, on pourra lire le contenu d'une position d'affichage par PEEK. Rappelons en efet que chaque caractère du jeu possède un code compris entre 32 et 126, que l'on nomme "code ASCII" et dont la liste complète est donnée à l'appendice 1.

La mémoire d'écran va de l'adresse #BB80 (48000) qui représente la première case en haut à gauche, à l'adresse #BFE0 (49120), dernière case en bas à droite. Nous avons bien 49120-48000=1120 octets.

Les adresses sont associées aux lignes/colonnes selon le schéma suivant:

Colonne / *Ligne*	0	1	2	3	37	38	39
0	48000	48001	48002	48003	48037	48038	48039
1	48040	48041	48042	48043	48077	48078	48079
2	48080	48081	48082	48083	48117	48118	48119
25	49000	49001	49002	49003	49037	49038	49039
26	49040	49041	49042	49043	49077	49078	49079
27	49080	49081	49082	49083	49117	49118	49119

Nous voyons que toutes les adresses se suivent de haut en bas et de gauche à droite. Trois formules simples permettent de convertir des coordonnées en adresse et vice-versa (A=Adresse, L=Ligne et C=Colonne), considérant que la première ligne et colonne ont le N° 0:

```
A=48000+C+L*40
L=INT((A-48000)/40)
C=A-INT(A/40)*40
```

3.1.1. Ligne et colonnes réservées

Vous avez certainement remarqué que sur le schéma, la première ligne (N° 0) et les deux colonnes de gauche sont séparées des autres par un trait gras: Ces positions sont en effet "interdites" à l'affichage en temps normal, la première ligne étant utilisée pour les messages de l'ordinateur, en particulier au cours des transferts de fichiers vers le K7. On peut toutefois afficher des caractères dans cette ligne en les plaçant un à un avec POKE, mais un moyen beaucoup plus élégant (et rapide !) de le faire est donné au paragraphe 3.5.2.

Quant aux deux colonnes de gauche, la première contient un code déterminant la couleur du fond pour toute la ligne, alors que le code placé dans la deuxième détermine la couleur des caractères affichés, toujours pour toute la ligne. Vous trouverez la liste complète de ces codes au paragraphe 3.4.

3.2. Ecrire avec PRINT – Les codes de contrôle

La "position d'affichage", dont nous avons parlé précédemment, est constamment matérialisé à l'écran par un carré noir clignotant: C'est le curseur, qui nous indique à quel endroit s'imprimera le prochain caractère. L'instruction PRINT est un des moyens d'effectuer cette impression.

Cette instruction, qui peut être remplacée par ? (point d'interrogation) nous permet bien sûr d'écrire toutes sortes de caractères "visibles" à l'écran, soit en les mentionnant directement, par exemple PRINT" Bonjour", soit un par un en les appelant par leur code, à l'aide de la fonction CHR$:

PRINT CHR$(65) imprimera un "A" (code 65) à la position du curseur. Évidemment, CHR$ n'est que de peu d'utilité pour écrire des caractères "visibles", comme les lettres ou nombres. Mais il en est autrement lorsqu'il s'agit d'imprimer des **codes de contrôle**.

3.2.1. Les codes de contrôle

En effet, nous avons vu que les codes du jeu de caractères étaient compris entre 32 et 126. Mais les valeurs de 0 à 31 sont également utilisées, simplement elles ne génèrent pas un caractère mais une **fonction**, qui concerne en général l'écran ou le clavier. Ceux-ci sont ainsi appelés car on les obtient en mode direct en actionnant simultanément la touche CTRL (CONTROL) et un lettre de l'alphabet.

La correspondance entre le mode direct et le mode programme (code à utiliser avec PRINT CHR$) est simple :

Donc :

Code CTRL=Code ASCII–64

CTRL A = CHR$(1), CTRL B = CHR$(2), etc...

Vous trouverez ci-dessous la liste complète des codes CTRL, en mode direct et mode programme. Souvent, les codes les plus importants sont générés par une touche spécialisée du clavier ; dans ce cas, cette touche est rappelée en dessous du code correspondant. D'autre part, il est normal que certains caractères soient omis car tous ne sont pas nécessairement utilisés.

MODES PROG	MODES DIRECT (CTRL)	FONCTION
CHR$(1)	CTRL A	Incrémente d'un caractère le buffer de ligne (voir éditeur, § 3.2.2).
CHR$(3)	CTRL C	Interromp l'exécution du programme (BREAK).
CHR$(4)	CTRL D	Démarre/arrête l'impression en double hauteur (cf. § 3.3).
CHR$(6)	CTRL F	Démarre/arrête la répétition sonore du clavier (bascule).
CHR$(7)	CTRL G	Sonnerie clavier.
CHR$(8)	CTRL H ou ←	Déplace le curseur d'une colonne vers la gauche.

CHR$(9)	CTRL I ou →	Déplace le curseur d'une colonne vers la droite.
CHR$(10)	CTRL J ou ↓	Déplace le curseur à la ligne suivante, même colonne.
CHR$(11)	CTRL K ou ↑	Déplace le curseur à la ligne précédente, même colonne.
CHR$(12)	CTRL L	Effacement de l'écran (également CLS en mode programme).
CHR$(13)	CTRL M ou RETURN	Retour chariot (valide la ligne en cours).
CHR$(14)	CTRL N	Efface la ligne où se trouve le curseur de l'écran, mais pas du buffer de ligne.
CHR$(16)	CTRL P	Théoriquement, recopie sur l'imprimante chaque ligne validée à l'écran par RETURN, mais actuellement sans fonction.
CHR$(17)	CTRL Q	Rend le curseur visible/invisible (bascule).
CHR$(19)	CTRL S	Bloque/débloque l'affichage (bascule).
CHR$(20)	CTRL T	Bloque/débloque les majuscules (bascule).
CHR$(24)	CTRL X	Interrompt l'édition d'une ligne (voir éditeur, § 3.2.2).
CHR$(27)	CTRL [ou ESC	Caractère Escape (cf. § 3.3).
CHR$(29)	CTRL]	Déblocage/blocage des deux colonnes de gauche (résultat visible : affichage en vidéo inverse).
CHR$(30)		Place le curseur à la première ligne, première colonne de l'écran HOME — accessible par programme uniquement.
CHR$(127)	DEL	Efface de l'écran et du buffer de ligne le dernier caractère entré, et déplace le curseur d'une colonne vers la gauche.

Déplacements du curseur à l'aide de PRINT

La commande PRINT à elle seule n'autorise pas un positionnement du curseur à n'importe quel endroit de l'écran adressage direct curseur, souvent PRINT AT ou PRINT @), mais des déplacements case par case sont possibles grâce aux codes 8 à 11, correspondant aux quatre flèches de l'écran.

Exemples :

```
FOR N=1 TO 5:?CHR$(11);:NEXT
```

Place le curseur cinq lignes plus haut;

```
FOR N=1 TO 4:?CHR$(10)CHR$(8);:NEXT
```

Déplace le curseur de quatre colonnes à gauche et quatre lignes plus bas.

3.2.2. L'éditeur de ligne

L'ensemble des commandes permettant d'écrire, puis de corriger un programme sont regroupées sous le nom d'"éditeur". On ne peut pas dire qu'Oric nous gâte de ce côté-là, puisque seules deux commandes et une instruction sont prévues. Mais à elles seules, elles autorisent toutes les modifications souhaitées.

L'éditeur de l'ORIC est appelé un éditeur de ligne, car il ne prend en compte qu'une ligne à la fois par opposition à un éditeur d'écran capable de gérer des groupes de lignes ou même tout un programme. Par contre, il gère l'affichage beaucoup plus rapidement qu'un éditeur d'écran. Son principe est le suivant:

— Chaque caractère frappé au clavier est affiché à l'écran, et en même temps placé dans une mémoire-tampon appelée "buffer".

— Les **quatre flèches** de déplacement, si elles ont un effet sur l'écran, n'en ont aucun sur le buffer.

— La touche DEL élimine du buffer le dernier caractère entré. Elle aura le même effet sur l'écran *uniquement si le curseur n'a pas été déplacé entre-temps avec les flèches* ! Prudence, donc.

— La touche **RETURN** vide le buffer, en "enregistrant" la ligne qui s'y trouve : Exécution en mode direct si la ligne est correcte sinon émission d'un message d'erreur, ou validation de celle-ci en tant que ligne de programme *si le premier caractère du buffer est un nombre.*

Si vous avez suivi jusqu'ici, vous n'aurez pas de problèmes pour la suite !

Modifications de lignes existantes

Deux codes CTRL sont disponibles à cet effet :

— CTRL A : Le fait de passer sur un caractère affiché à l'écran avec CTRL A recopie celui-ci dans le buffer ; on peut ainsi recopier des lignes entières en passant dessus, sans avoir à les retaper. Évidemment, on peut recopier une ligne jusqu'à un certain point, taper ensuite au clavier des caractères à modifier, puis continuer à recopier cette ligne ou une autre à partir de n'importe quel point. N'oublions pas que les flèches, si elles déplacent le curseur, n'ont *aucun effet* sur le buffer.

— CTRL X : Affiche un \ (backslash) à la position du curseur, et vide le buffer. Le contenu de celui-ci n'est alors pas pris en compte, contrairement à ce qui se passe avec RETURN.

Exemples :

Pour recopier une ligne de programme en changeant son numéro :

— Taper EDIT <Numéro de la ligne>

— Le curseur est placé juste avant le numéro. Retaper celui-ci et s'il est plus long que l'ancien monter d'une ligne avec ↑ , puis taper les chiffres restants.

— Placer le curseur avec les flèches sur le premier caractère de la ligne.

— Copier toute la ligne, en faisant passer le curseur dessus avec CTRL A.

— Valider avec RETURN.

Pour insérer dans une ligne de programme :

— Lister la ligne avec EDIT<Numéro de ligne>

— Faire passer le curseur sur la ligne avec CTRL A, jusqu'à amener le curseur sur le caractère *suivant le point d'insertion.*

— Faire monter le curseur d'une ligne avec ↑ , puis taper le texte à insérer.

— Revenir sur le caractère suivant le point d'insertion avec les flèches.

— Copier la ligne jusqu'à la fin avec CTRL A, puis valider par RETURN.

Ces deux exemples devraient vous permettre de faire toutes sortes de modifications dans vos programmes, et même dans des lignes en mode direct. Rappelez-vous que :

— Une ligne est éliminée d'un programme simplement en tapant son numéro.

— Il faut faire attention lorsque vous tapez des nombres à l'écran : Ils seront considérés comme des numéros de ligne s'ils sont suivis de RETURN, et en effaceront éventuellement de votre programme.

— Tous les codes CTRL sont ignorés par le buffer, excepté bien sûr CTRL A et CTRL X : Vous pouvez non seulement vous déplacer sans risques, mais aussi passer en minuscules ou effacer de l'écran une ligne qui vous gêne.

— Lorsque l'écran est vide, il est rempli d'espaces. Quand vous taper CTRL A sur une ligne vide, vous ne faites pas "rien", vous mettez des espaces (code ASCII 32) dans le buffer !

3.3. *Utilisation de ESCAPE*

Déplacez le curseur de deux ou trois colonnes vers la droite, puis tapez la touche ESC ; ensuite, tapez la lettre "L", puis un peu de texte : Vous voyez l'effet à l'écran d'un code ESCAPE, en l'occurence ESC L. Ils fonctionnent de la manière suivante :

Le caractère "Escape" (code ASCII 27) indique à l'ORIC que le caractère suivant ne doit pas être affiché, mais considéré comme un attribut ; cet attribut sera valide sur le restant de la ligne où il se trouve, mais pas sur les suivantes. Il existe toutes sortes d'attributs, déterminant la couleur du fond et des caractères, leur clignotement, le passage en double taille ou le changement de jeu. Vous trouverez la liste complète des attributs au § 3.4.

Nous avons vu qu'il est extrêmement simple d'utiliser Escape en mode direct ; ce n'est pas plus compliqué par programme, il suffit de taper :

```
PRINT CHR$(27)"<caractère> "
```

ou bien :

```
PRINT CHR$(27)CHR$(<code caractère>)
```

Exemples :

Tapez la séquence de caractères suivante ("ESC" est bien sûr la touche ESC) :

ESC Q ESC C JAUNE/ROUGE ESC L FLASH

Maintenant, tapez ce programme puis exécutez-le :

```
10 E$=CHR$(27)
20 ?E$"Q"E$"CJAUNE/ROUGE"E$"LFLASH"
```

— Vous constaterez que l'effet est identique à celui de la ligne précédente en mode direct.

— A la ligne 10, on a mis le caractère Escape dans la variable E$, pour s'éviter de taper trois fois de suite CHR$(27).

— Les lettres "C" et "L", incluses dans les messages, ne sont pas considérées comme des caractères à imprimer mais comme des attributs, car elles suivent immédiatement le caractère Escape.

Utilisation de la double hauteur

Vous aurez certainement remarqué, en essayant les codes CTRL, que CTRL D provoquait un dédoublement de l'impression sur deux lignes. D'autre part, ESC J imprime un caractère bizarre qui en fait est la moitié (inférieure ou supérieure, selon la ligne où l'on se trouve) d'un caractère complet en double hauteur. C'est la réunion de ces deux fonctions qui permettra l'affichage de caractères en double taille. Mais attention : Les lignes paires ne peuvent contenir que la moitié supérieure, et les lignes impaires que la moitié inférieure d'un caractère.

Exemple :

```
10 E$=CHR$(27):CLS:?CHR$(4)
20 ?"    "E$"JDOUBLE"E$"NDOUBLE FLASH"
30 ?CHR$(4)
```

REMARQUE : CTRL D ou CHR$(4) agissant comme une bascule, il faut être sûr que la double taille n'est pas déjà en fonction quand on l'utilise. Il est

cependant facile de forcer la double ou simple taille avec POKE 618 (cf. § 2.3.3); dans ce cas, les lignes 10 et 30 deviendraient:

```
10 E$=CHR$(27):CLS:?:POKE 618,67
et
30 POKE 618,3
```

D'autre part, deux espaces ont été inclus à la ligne 20, juste avant Escape, ce qui n'était pas le cas dans le programme précédent. La raison en est très simple: La première colonne est dévolue au code déterminant la couleur du fond, et ne peut donc accepter que des caractères Escape ayant le même effet. C'est aussi le cas pour la deuxième colonne, qui elle détermine la couleur des caractères. Si un code de type différent est imprimé dans ces colonnes, en général le fond et l'affichage deviennent noirs.

3.3.1. La Hardcopy texte

Une "hardcopy"est une copie exacte sur imprimante de ce qui apparaît à l'écran. Le programme qui suit exécute cette fonction pour l'écran texte (Un programme analogue effectuant la hardcopy graphique est donné au § 4.4), et il est numéroté à partir de 60000 afin d'être inclus comme sous-programme dans vos applications personnelles. Ne comprenant aucun code spécifique, cette routine fonctionne sur toutes les imprimanes.

```
60000 REM  HARDCOPY TEXTE
60002 REM
60010 AD=48000:AF=49080:LPRINT TAB(20):CALL#ED01
60020 FOR N=AD TO AF STEP 40
60030 FOR A=0TO 39
60040 D=PEEK(A+N):IF D<32 OR D>126 THEN D=32
60050 LPRINT CHR$(D);
60060 NEXT A:LPRINT:LPRINT TAB(20);
60070 NEXT N:CALL#E804
```

Explications:

— 60010: Fixe les limites de la zone à copier (écran), et centre celle-ci sur une imprimante 80 col. La routine appelée désactive le clavier pour éviter le interférences pendant la durée de l'impression.

— 60040: Lit le contenu de chaque adresse-écran, et remplace les attributs non imprimables par des espaces (ASCII 32).

— 60050: Envoie le code du caractère lu à l'imprimante.

— 60060: Saut de ligne tous les 40 caractères, et recentrage de la ligne suivante.

— 60070: Réinitialisation du clavier.

Notez que les commandes LPRINT TAB(X) ne présentent d'intérêt que dans le cas d'une imprimante 80 colonnes, pour le centrage du texte qui n'en fait que 40. Elles devront bien sûr être supprimées si vous utilisez un modèle 40 colonnes.

3.4. *L'instruction PLOT – Les attributs d'écran*

Cette instruction, dont l'effet surprend un peu ceux qui sont habitués à d'autres machines, est néanmoins extrêmement puissante:

— Elle permet d'afficher à l'écran des variables ou constantes de chaînes, ainsi que des attributs ou des caractères isolés appelés par leur code.

— Elle ne permet *pas* d'imprimer des variables ou constantes numériques ou des codes CTRL, et ne *déplace pas le curseur.*

— L'affichage de nombres reste néanmoins possible, à condition de les convertir en chaîne au préalable à l'aide de STR$: Si on veut imprimer en col. 5 ligne 10 la variable N, nous devrons écrire: PLOT 5,10,STR$(N).

— On peut imprimer un caractère du jeu ASCII en vidéo inverse en écrivant:

PLOT<Col.>,<Ligne>,<Code Car.>+128.

(Le jeu ASCII complet avec ses codes est donné à l'Appendice 1).

Exemples: Tapez au clavier:

```
PLOT 10,10,22
```

Et le petit programme suivant:

```
10 CLS:?
20 ?"JAUNE/ROUGE  FLASH"
30 PLOT 0,1,17:PLOT 1,1,3:PLOT 13,1,12
```

Comme on peut le voir dans cet exemple, PLOT sert non seulement à afficher des caractères mais aussi, comme ESC, à placer des attributs à l'écran. En fait, chaque caractère considéré comme attribut après ESC possède un équivalent numérique utilisable par PLOT.

Il ne faut *absolument pas confondre ces attributs avec les codes CTRL, qui ont pourtant les mêmes nombres* (0 à 31). Ceux-ci s'emploient avec PRINT, et correspondent à un standard international (ASCII), alors que les attributs sont des codes internes de l'ORIC, affichés par PLOT ou POKE.

Attributs d'écran

Pour chacun d'eux nous donnons la valeur à donner derrière PLOT ou POKE, la séquence ESCAPE correspondante et la fonction exacte:

Couleurs affichage

0	ESC @	Noir
1	ESC A	Rouge
2	ESC B	Vert
3	ESC C	Jaune
4	ESC D	Bleu
5	ESC E	Magenta
6	ESC F	Cyan
7	ESC G	Blanc

Taille et jeu de caractères

8	ESC H	Taille normale, jeu normal (ASCII)
9	ESC I	Taille normale, jeu graphique
10	ESC J	Double taille, jeu normal
11	ESC K	Double taille, jeu graphique

Clignotement

12	ESC L	Clignotant, taille normale, jeu normal
13	ESC M	Clignotant, taille normale, jeu graphique
14	ESC N	Clignotant, Double taille, jeu normal
15	ESC O	Clignotant, Double taille, jeu graphique

Couleur de fond

16	ESC P	Noir
17	ESC Q	Rouge
18	ESC R	Vert
19	ESC S	Jaune
20	ESC T	Bleu
21	ESC U	Magenta
22	ESC V	Cyan
23	ESC W	Blanc

— On peut constater que les codes sont organisés de façon logique, et que les couleurs du fond correspondent à celles de l'affichage plus 16.

— En ce qui concerne les caractères double taille, l'utilisation de PLOT est assez malaisée: comme ceux-ci occupent deux lignes à l'écran, il faudrait placer le code par PLOT sur les deux lignes concernées pour obtenir le résultat souhaité ! Escape est, dans ce cas, beaucoup plus pratique d'emploi.

N'oublions pas enfin que PLOT peut aussi servir à écrire n'importe quelle chaîne de caractères n'importe où à l'écran (une à la fois cependant), et se révèle donc très utile pour imprimer des messages ou animer des objets rapidement et sans déplacer le curseur.

3.4.1. Le programme "Tennis"

Le programme suivant montre comment on peut exploiter les particularités de PLOT dans un jeu en temps réel, en l'occurrence un jeu de tennis à un joueur rappelant les ancêtres (déjà !) de nos actuels jeux de café.

Peu d'explications sont nécessaires: On peut choisir la taille du terrain, la barre d'espacement envoie une balle, la raquette se déplace avec les flèches ↓ ↑ et la touche R permet de changer la taille du terrain en cours de partie.

```
 10 REM↓↑TENNIS
 12 REM
 50 R=1:X=1:Y=1:FX=1:FY=1:AX=8:S=0
100 GOSUB 9000
```

```
 110 GOSUB 6000
 120 GOSUB 8000
 130 GOSUB 7000
 140 POKE 618,10
 150 PLOT4,1,"SCORE: 0    ":PLOTX,Y,64
 998 REM
1000 REM  BOUCLE PRINCIPALE
1002 REM
1100 X$=KEY$:IF X$<> "" THEN AX=ASC(X$)
1110 ON AX-7 GOTO 1150,1150,1130,1140
1120 GOTO 1150
1130 IF R<23 THEN PLOT1,R,96:R=R+1:PLOT1,R,126:PLOT1,R+1,126
1135 GOTO 1150
1140 IF R>3 THEN PLOT1,R+1,96:R=R-1:PLOT1,R,126:PLOT1,R+1,126
1150 IF Y<4 THEN FY=1:SHOOT:PLAY0,0,0,0
1160 IF Y>23 THEN FY=-1:SHOOT:PLAY0,0,0,0
1170 IF X>T THEN FX=-1:SHOOT:PLAY0,0,0,0
1200 IF X>2 THEN 1300
1210 IF X<2 THEN 1500
1220 IF Y=R THEN FX=1:FY=-1:SHOOT:PLAY0,0,0,0:S=S+1:IF Y=3 THEN
     FY=1
1230 IF Y=R+1 THEN FX=1:FY=1:SHOOT:PLAY0,0,0,0:S=S+1:IF Y=24 THEN
     FY=-1
1240 PLOT10,1,STR$(S)
1300 PLOTX,Y,32:X=X+FX:Y=Y+FY:PLOTX,Y,64
1310 GOTO 1100
1498 REM
1500 REM  BALLE PERDUE
1502 REM
1510 ZAP:S=0:AX=32
1520 GET X$:IF X$="R" THEN RUN
1530 IF X$ <>CHR$(32) THEN 1520
1540 IF Y=R OR Y=R+1 THEN PLOT1,Y,126 ELSE PLOT1,Y,96
1550 GOTO 130
5998 REM
6000 REM  CHOIX DU TERRAIN
6002 REM
6010 CLS:POKE 618,67:POKE 616,5:?
6020 ?SPC(12)CHR$(27)"JTENNIS"
6030 POKE 618,3:POKE 616,10:?:INPUT"TAILLE DU TERRAIN (0 A 14)";T
6040 IF T<0 OR T>14 THEN 6000
6050 T=T+20:RETURN
6998 REM
7000 REM  SERVICE BALLE
7002 REM
7010 IF RND(1)<0.5 THEN FY=-1 else FY=1
7020 X=T+1:FX=-1
7030 Y=INT(RND(1)*21)+3:RETURN
7998 REM
```

```
8000 REM  DESSIN TERRAIN
8002 REM
8005 CLS
8010 FOR N=48122 TO 49042 STEP 40
8020 POKE N,96:POKE N+T+1,128:NEXT
8030 R=13:PLOT1,13,126:PLOT1,14,126
8040 AD=48122:Z=0
8050 FOR N=0 TO T STEP 2
8060 DOKE AD+N,32896:NEXT
8070 IF Z=0 THEN Z=1:AD=49042:GOTO 8050
8080 RETURN
8998 REM
9000 REM  INITIALISATIONS
9002 REM
9010 READ AD:IF AD=0 THEN 9100
9020 FOR N=0 TO 7
9030 READ DA:POKE AD+N,DA
9040 NEXT
9050 GOTO 9010
9100 RETURN
9500 DATA 46592,12,30,63,63,63,30,12,0
9510 DATA 47088,31,31,31,31,31,31,31,31
9520 DATA 46848,0,4,4,4,4,4,4,0
9600 DATA 0
```

Explications

50 :Attribue le plus tôt possible les variables importantes pour la vitesse d'exécution.
R=Ordonnée de la raquette
X=Abscisse de la balle
Y=Ordonnée de la balle
FX=Valeur à ajouter à l'abscisse de la balle (1 ou −1)
FY=Valeur à ajouter à l'ordonnée de la balle (1 ou −1)
S=Score

110-130 :Appel des sous-programmes de définition des caractères spéciaux, saisie de la taille du terrain, dessin du terrain et service.

9000-9600 :Dessin de la balle (code 64), de la raquette (code 126) et du pointillé servant de fond à la raquette (96). (Cf. § 3.7.3).

6000-6050 :En-tête du jeu et saisie de la taille du terrain (T).

8000-8080 :Tracé du terrain. Le caractère 128 (carré noir) est placé par POKE et PLOT aux emplacements corrects.

7000-7040 :Service aléatoire. On détermine d'abord un sens aléatoire pour la balle (7020), puis une ordonnée aléatoire pour son apparition.

1100	:Saisie de la touche pressée.
1130-1140	:Gestion des mouvements de la raquette.
1150-1170	:Détermination du sens de rebond de la balle, en fonction de son angle d'impact.
1200	:Teste si la balle atteint l'abscisse de la raquette.
1210	:Teste si la balle a été perdue.
1220-1230	:Balle rattrapée; gestion de son rebond sur la raquette.
1240	:Affichage du nouveau score.
1300	:Efface la balle à ses anciennes coordonnées, et l'affiche aux nouvelles.
1310	:Retour dans la boucle.

3.5. Extensions au Basic – Le langage-machine

Dans les chapitres précédents, nous avons passé en revue toutes les commandes et fonctions offertes par le Basic pour la gestion de l'affichage, en rapport avec PRINT et PLOT. On peut cependant aller nettement plus loin dans les possibilités et le confort d'utilisation, en exploitant notamment les variables-système liées à l'écran.

3.5.1. Ecrire et lire à l'écran avec POKE et PEEK

Nous avons vu au paragraphe 3.1 que le stockage de tout le contenu de l'écran en mémoire permettait d'y afficher n'importe quel caractère avec POKE, en tapant POKE<Adresse>, <Code Car.> (les codes ASCII de tous les caractères se trouvent à l'appendice 1). En général, il est beaucoup plus pratique d'utiliser PLOT qui a la même fonction en étant plus "parlant", dans la mesure où la position d'affichage y est donnée non pas par une adresse mais par des numéros de ligne et colonne.

L'usage de POKE se révèle cependant indispensable dans le cas où l'on veut accéder à la première ligne de l'écran, interdite à PRINT et PLOT. Il suffira d'écrire :

```
10 M$=" <Message à imprimer> "
20 FOR N=1 TO LEN(M$)
30 POKE 48001+N+ <Col.début> ,ASC(MID$(M$,N,1))
40 NEXT N
```

Toute chaîne de caractères attribuée à M$ à la ligne 10 sera imprimée sur la première ligne de l'écran, à partir de <Col.début>. Une autre méthode, encore plus rapide, est donnée au paragraphe 3.5.2.

En ce qui concerne les codes fournis comme arguments à POKE, tous ceux compris entre 32 et 128 sont considérés comme des caractères du jeu ASCII, et ceux compris entre 0 et 31 comme des *attributs* ; les règles sont identiques à celles en vigueur pour PLOT, et exposées plus haut. De même, un caractère sera imprimé en vidéo inverse si on POKE son code +128.

Enfin, en lisant un code dans une adresse de l'écran, PEEK établit la même correspondance avec la fonction SCRN que POKE avec PLOT. Rappelons que SCRN ne permet pas plus de lire le contenu de la première ligne que PLOT ne permet d'y écrire, PEEK sera donc indispensable dans ce cas.

3.5.2. L'utilisation des variables-système

Modification de la taille de l'écran

Trois variables déterminent entièrement les limites haute et basse de la zone d'évolution du curseur :

— *L'adresse de début de la première ligne de l'écran* est contenue dans #26D,#26E (621,622). Normalement, cette valeur est de 48000, laissant la première ligne (qui commence à 47960) hors d'atteinte du curseur. Pour récupérer celle-ci, il suffira donc de taper DOKE 621,47960. Effacez l'écran, et voyez où se place le curseur !

Maintenant faites un RESET, écrivez un peu de texte à l'écran puis entrez la ligne suivante en mode direct :

```
DOKE621,47960:?CHR$(30)"CE MESSAGE EST INAMOVIBLE!":DOKE621,
48000
```

Vous constatez que votre message, qui ne peut plus être effacé, s'est affiché considérablement plus vite qu'avec POKE, et en utilisant tout bêtement PRINT. Rappelons que CHR$(30) place le curseur au coin supérieur gauche de l'écran, sans effacer celui-ci.

Inversement, vous pouvez interdire au curseur un nombre quelconque de lignes, avec la formule :

DOKE 621,<Nombre de lignes interdites>*40+47960

Seuls, PLOT et POKE permettront d'aller encore dans ces lignes. Évidemment, rien n'est plus simple que de redimensionner l'écran à sa taille maximale, l'effacer puis redéfinir ses limites tout de suite après : Le tout est extrêmement rapide.

— *La dernière ligne de l'écran* est déterminée par #26F (623). Cette valeur, qui est normalement de 27, est elle aussi modifiable par l'utilisateur selon la formule :

POKE 623,<N° de la ligne inférieure>

Plus qu'un long discours, le petit programme suivant vous donnera un aperçu des possibilités offertes :

```
10 CLS:PLOT3,20,"VOUS NE POUVEZ PAS ECRIRE ICI"
20 POKE 623,15
30 FOR N=1 TO 20
40 ?:?SPC(10)"TEXTE....."":WAIT15
50 ?SPC(10)"A IMPRIMER":WAIT15:NEXT
60 CLS:PLOT6,22,"NI EFFACER D'AILLEURS!"
```

Pour terminer, il faut savoir que :

— PLOT et POKE peuvent toujours accéder aux lignes hors limites en bas de l'écran.

— Si vous changez à la fois les limites haute et basse, commencez par le bas, ou alors comptez le nombre de lignes données comme argument à POKE 623 à partir de la *nouvelle* limite supérieure.

— Rien ne vous oblige à donner une adresse multiple de 40 (47960, 48000, 48040, etc.) comme argument à DOKE 621. Mais dans ce cas, considérez-vous comme seul responsable de ce que vous verrez à l'écran ! Même remarque pour des nombres supérieurs à 27 dans POKE 623.

L'adressage direct du curseur

Les deux variables #268 (616) et #269 (617) contiennent respectivement la ligne et la colonne où se trouve le curseur. L'instruction PRINT AT ou PRINT @ , bien connue sur d'autres machines, peut se simuler grâce à ces variables selon la formule :

POKE 616,< N° ligne >–1:PRINT:POKE 617,< N° Col. >

Le PRINT est indispensable après POKE 616, car le positionnement vertical du curseur n'est effectif qu'après cette instruction (Cf. § 2.3.3).

Exemple :

```
10 CLS:CO=30
20 FOR LI=27 TO 1 STEP–1
30 POKE616,LI–1:PRINT:POKE617,CO
40 PRINT"LIGNE"LI;:CO=CO–1:WAIT10
50 NEXT:POKE616,22:PRINT
```

Nous constatons que toutes les opérations d'affichage de ce programme sont effectuées par PRINT, POKE n'étant utilisé que pour les déplacements curseur.

Une autre variable gère également la ligne où se trouve le curseur : Il s'agit de #12,#13 (18,19), contenant l'adresse de début du Numéro de ligne où se trouve le curseur. Contrairement à POKE 616 toutefois, DOKE18,<Adresse> est immédiatement effectif et ne nécessite pas de PRINT à la suite.

REMARQUE :

— Bien entendu, ces variables peuvent aussi nous retourner des valeurs, étant constamment réactualisées par ORIC en cours de fonctionnement. La position exacte du curseur (ligne-colonne) sera retournée par : ?PEEK(616)PEEK(617).

— Souvent en utilisant 616 ou 18,19 des curseurs "parasites" seront imprimés, surtout si des ; sont utilisés après PRINT. Le remède est simple : Faites disparaître le curseur avec CTRL Q, PRINT CHR$(17) ou POKE 618,2 (ce n'est pas les moyens qui manquent !).

3.6. *Modes texte et LORES*

3.6.1. Le mode LORES

Celui-ci ne présente que peu de différences avec le mode Texte normal, il est mis en service par:

LORES 0, qui est identique à TEXT avec un affichage en vidéo inverse, et utilise le jeu de caractères normal.

LORES 1, comme ci-dessus mais utilisant le deuxième jeu de caractères (mosaïques graphiques) sur tout l'écran.

REMARQUE:

— Ces deux fonctions sont obtenues en plaçant le code du jeu désiré (8 ou 9) dans la première colonne réservée, puis de remplir la deuxième colonne avec le code "fond noir". Le fait de fournir un code de couleur de fond à la deuxième colonne (normalement réservée à la couleur des caractères) provoque l'affichage en blanc sur la couleur choisie.

— En conséquence de ce qui vient d'être dit, il est impossible de changer les couleurs de l'écran LORES par PAPER ou INK, ceux-ci agissant précisément sur les deux premières colonnes.

— Les couleurs valides en mode texte sont mémorisées dans #26B (619) pour le fond et #26C (620) pour l'affichage. Si vous effacez l'écran en mode LORES, ces valeurs seront restituées ainsi que le jeu de caractères standard, vous faisant retourner au mode normal. Heureusement, il existe un moyen très simple pour résoudre ces problèmes:

3.6.2. Choix d'attributs pour tout l'écran

Les deux variables que nous avons mentionnées (619,620) offrent en quelque sorte un LORES "à la carte": On peut choisir n'importe quelle couleur de fond et de premier plan, ainsi que n'importe quel autre attribut d'affichage instantanément valide sur *tout l'écran*: Caractères double taille, clignotants, changement de jeu etc... (La liste complète des attributs est donnée au § 3.4). Il suffit de taper:

```
POKE619,<Attribut1> :POKE620,<Attribut2>
```

Et de respecter les quelques consignes suivantes:

— L'astuce consiste à ne *pas* placer l'attribut correct dans la colonne adéquate (couleur fond dans 619 et premier plan dans 620). Si c'était le cas, PAPER et INK sont bien sûr plus intéressants.

— Les nouvelles valeurs entrent en vigueur après effacement de l'écran. Le fait d'effacer l'écran ensuite respectera ces valeurs, contrairement à ce qui se passe en LORES.

— Étant donné que l'attribution aux colonnes réservées d'un code qui ne leur est pas destiné rend le fond noir et les caractères blancs, l'affichage se fera toujours en clair sur sombre.

Exemples:

Simulation de LORES 1:

```
POKE619,9:POKE620,16:CLS
```

Blanc sur rouge, caractères géants:

```
POKE619,10:POKE620,17:CLS puis CTRL D
```

Rouge sur noir, clignotant:

```
POKE619,1:POKE620,12:CLS
```

3.7. *Les jeux de caractères*

3.7.1. Généralités

Ceux-ci sont au nombre de deux:

— Le jeu standard, correspondant à la norme américaine ASCII et comprenant les caractères alphanumériques et les ponctuations. Les codes de ces caractères vont de 32 ("Espace") à 126 (zone grisée). La liste du jeu ASCII avec tous ses codes se trouve à l'Appendice 1.

— Le deuxième jeu, composé de mosaïques graphiques, comprenant 64 caractères dont les codes vont de 32 à 95. Ces caractères s'obtiennent par l'écriture d'attributs appropriés à l'écran (voir plus haut).

3.7.2. Principes de codage

Les caractères sont dessinés dans une matrice de 8×8, chacun des points de la matrice pouvant être "allumé" (noir) ou "éteint" (blanc). Un point est représenté par un Bit, il faut donc un octet pour stocker 8 points ou une ligne horizontale de la matrice, et 8 octets pour stocker un caractère en mémoire. Ces 8 octets sont situés en mémoire dans huit adresses consécutives, l'ensemble de ceux-ci constituant deux zones: Entre les adresses #B400 (46080) et #B7FF (47103) pour le jeu standard et #B800 (47104)–#BB57 (47959) pour le deuxième jeu.

Étant donné que ces zones sont en mémoire vive, n'importe quel caractère peut être entièrement redessiné par l'utilisateur en fonction de ses besoins, en modifiant la valeur des adresses le définissant: Minuscules accentuées, invaders, etc.

La première des huit adresses codant un caractère donné est trouvée par la formule:

<Adresse>=46080+8*<code ASCII du car.> pour le jeu normal, et
<Adresse>=47104+8*<code ASCII du car.> pour le jeu graphique.

L'exemple suivant montre comment un caractère est stocké dans ces adresses:

Matrice						*Adresses*	*Valeur décimale*	*Valeur binaire*
Bits	32	16	8	4	2	1			
			*				46600	8	00 001000
		*		*			46601	20	00 010100
	*				*		46602	34	00 100010
	*				*		46603	34	00 100010
	*	*	*	*	*		46604	62	00 111110
	*				*		46605	34	00 100010
	*				*		46606	34	00 100010
							46607	0	00 000000

Vous remarquerez que les bits de poids 64 et 128 sont inutilisés, ils ont d'ailleurs été omis de la matrice pour plus de clarté. En fait, seuls les six

premiers bits d'un octet sont utilisés pour le dessin, la matrice fait donc 8×6 cases. Si vous avez du mal à comprendre comment le contenu de la matrice est codé, voyez aussi paragraphe 2.1.).

3.7.3. Comment créer un caractère

Pour y arriver facilement, commencez par suivre la méthode ci-dessous:

— Dessinez sur une feuille de papier un damier de six colonnes et huit lignes.

— Au-dessus de chaque colonne, marquez la valeur de chaque Bit, comme pour la matrice du "A" ci-dessus (32,16,8,4,2 et 1).

— Dessinez votre caractère en noircissant les cases appropriées.

— A droite à côté de chaque ligne, notez le nombre résultant de l'addition des valeurs de toutes les cases noires de la ligne, cette valeur étant notée au-dessus de chaque colonne. (Une case blanche vaut 0).

— Choisissez le caractère à remplacer par le vôtre, puis repérez son code ou demandez-le à ORIC en tapant: ?ASC("<Caractère>").

— Trouvez la première adresse des 8 codant le caractère avec <Adresse>=46080+<code ASCII>*8.

— Placer la valeur marquée à côté de la première ligne du damier dans l'adresse trouvée, puis la valeur à côté de la deuxième ligne dans l'adresse trouvée +1, etc. jusqu'à la ligne 8 dans <Adresse>+7: POKE<Adresse>,<Valeur>.

Voilà ! votre caractère est redessiné, utilisez-le pour vérifier. Tout cela a l'air assez long, mais en fait l'habitude vient très vite.

Si vous aimez le calcul mental, voici une autre méthode qui économise des instructions, donc du temps et de la place. Nous avons vu qu'il fallait POKER huit nombres dans huit adresses. Mais pourquoi ne pas le faire sur quatre "couples" d'adresses avec DOKE ? Soient les huit adresses A0 à A7 et les huit valeurs V1 à V8, il faut écrire:

```
DOKE A0,V1+V2*256:DOKE A2,V3+V4*256:DOKE A4,V5+V6*256:DOKE A6,
V7+V8*256
```

Exemple :

Création d'un "invader" à la place du caractère " @ " (Code ASCII 64).

Matrice							*Adresses*	*Valeur décimale*	*Valeur binaire*
Bits	32	16	8	4	2	1			
			*	*			46592	12	00 001100
		*	*	*	*		46593	30	00 011110
	*		*	*		*	46594	45	00 101101
	*	*	*	*	*	*	46595	63	00 111111
		*	*	*	*		46596	30	00 011110
			*	*			46597	12	00 001100
		*	*	*	*		46598	30	00 011110
	*	*			*	*	46599	51	00 110011

Ces valeurs seront entrées à l'aide d'une des deux séquences suivantes:

1 :

```
10 POKE 46592,12:POKE 46593,30:POKE 46594,45:POKE 46595,63
20 POKE 46596,30:POKE 46597,12:POKE 46598,30:POKE 46599,51
```

2 :

```
10 DOKE 46592,7692:DOKE 46594,16173:DOKE 46596,3102:DOKE 46598,13086
```

3.8. *Les programmes d'applications: CARGEN et GENETEX*

Ces deux programmes reprennent à eux deux quasiment toutes les commandes de gestion de l'écran, de la mémoire ou des périphériques qui ont été vues jusqu'à présent. CARGEN est directement en rapport avec le chapitre précédent, quant à GENETEX il vous permettra, comme son nom l'indique, de faire du "traitement de textes".

3.8.1. Le programme "CARGEN"

Les procédures de modifications de caractères exposées ci-dessus sont idéales lorsque vous n'avez à en redessiner que quelques-uns pour une

application particulière. Mais si vous voulez reconfigurer un jeu complet, puis le sauvegarder sur cassette pour réutilisation ultérieure, voici ce que vous pourrez faire avec le programme CARGEN :

1) Afficher sur l'écran le jeu complet, ainsi qu'une matrice agrandie.
2) Choisir un caractère du jeu en se déplaçant dans celui-ci, avec un curseur spécial.
3) Transférer le caractère choisi dans la matrice, le modifier en dessinant avec les 4 flèches, puis replacer le caractère n'importe où dans le jeu.
4) Transférer des caractères d'un emplacement à l'autre dans le jeu.
5) Sauvegarder et relire un jeu sur cassette.

Commandes disponibles

<>	Déplacements dans le jeu de caractères.
1-2	Choix du jeu (1=Jeu standard, 2=Jeu graphique).
C	Copier dans la matrice le caractère pointé par le curseur "jeu".
RETURN	Copier à la position du jeu pointée par le curseur le caractère se trouvant dans la matrice.
Flèches	Déplacements dans la matrice.
ESPACE	Trace/Efface un point de la matrice (bascule).
DEL	Efface la matrice.
ESC	Permet d'accéder aux fonctions de Sauvegarde et Chargement d'un jeu, et abandon du programme.

```
~
1 REM       ** CARGEN **
2 REM
20 CLS
28 REM      GESTION POINTEURS
29 REM
30 IF A(1)THEN DOKE#9C,DEEK(#400):GOTO70ELSEDOKE#400,DEEK(#9C)
60 :GOSUB9100' Initialisations
70 EN$="CARGEN Generateur de caracteres "+CHR$(96)+" BVRP"
80 PRINTCHR$(17);
90 :GOSUB9000' En-tete
100 GOSUB8000' Dessin Jeu
```

```
105 GOSUB8200' Dessin grille
110 GOSUB8500' Aides
200 GOSUB7000' Gestion curseurs
210 GOTO3000' Sauveg. jeu+fin prog.
495 REM
1000 MS$="JEU No"+STR$(J)+"   NOM: "+NJ$
1010 PLOT5,9,MS$
1020 RETURN
1098 REM
2000 REM   LECTURE/ECRITURE CARACT.
2002 REM
2005 PLAY1,0,3,1000
2010 PRINTCHR$(17);
2020 PO=((XJ-3)/2)+YJ/40*18
2040 IFJ=1THENAA=#B500+PO*8
2050 IFJ=2THENAA=#B900+PO*8
2080 IFAS=13THEN2500
2098 REM
2100 REM   Lecture caractere
2102 REM
2110 NO(1)=127:NO(0)=32
2130 FORM=0TO280STEP40
2140 AW=AA+M/40
2145 Q1=PEEK(AW)/2:R=PEEK(AW)-(INT(Q1)*2)
2150 FORN=5TO0STEP-1
2170 IFQ1=.5THENPOKE48687+M+N,127:Q1=0ELSEPOKE48687+M+N,NO(R)
2175 R=Q1-(INT(Q1/2)*2):Q1=Q1/2
2180 NEXTN
2190 NEXTM
2200 PRINTCHR$(17);:PLAY0,0,0,0:RETURN
2498 REM
2500 REM   Ecriture caractere
2502 REM
2510 FORM=0TO280STEP40
2520 AW=AA+M/40:T=0
2540 FORN=0TO5
2550 IFPEEK(48687+M+(5-N))=127THENT=T+2^N
2560 NEXTN
2570 POKEAW,T
2580 NEXTM
2600 PRINTCHR$(17);:ZAP:RETURN
2998 REM
3000 REM   SAVE, LOAD, FIN PROG.
3002 REM
3010 CLS:EN$="CARGEN  Fonctions annexes       ":GOSUB9000
3020 XM=0:YM=0
3030 POKE616,6:PRINT
3040 PRINT"<S>ave: Sauvegarde sur cassette"
3050 PRINT:PRINT"<L>oad: Chargement"
```

```
3060 PRINT:PRINT"<R>etour au programme"
3070 PRINT:PRINT"<F>in du programme"
3100 GETX$
3110 IFX$="S"THEN3200
3120 IFX$="L"THEN3500
3130 IFX$<>"F"THEN3140
3135 EN$=CHR$(14):GOSUB9000:DOKE48000,1808:DOKE(#9C),1283:WAIT10:CA
LL#C000
3140 IFX$="R"THENCLS:GOTO70
3190 GOTO3100
3198 REM
3200 REM   Save
3202 REM
3240 PRINT:PRINTEA$"Nom du jeu ";E$"W";:INPUTNJ$
3250 PRINTCHR$(4);CHR$(17);
3260 PRINT:PRINT:PRINTE$"N     PREPAREZ LA CASSETTE":PRINT
3270 PRINTE$"N     ET PRESSEZ LA <BARRE>"
3280 PRINTCHR$(4)
3290 GETX$:IFX$<>" "THEN3000
3300 FORN=1TO5:PRINTCHR$(11);CHR$(14);:NEXTN
3310 PRINTCHR$(17);
3315 CSAVENJ$,A#B500,E#BB7F
3320 CLS:GOTO3000
3498 REM
3500 REM   Load
3502 REM
3510 CLS
3520 PRINT:PRINT:PRINTCHR$(4)
3530 PRINTE$"J"E$"B   ATTENTION !"
3540 PRINT:PRINT:PRINTCHR$(4)
3550 PRINT" Apres avoir charge un jeu de caracte-";
3560 PRINT"res a partir de la cassette, vous etes";
3570 PRINT"en mode direct.":PRINT
3580 PRINT" Tapez"E$"AGOTO 1"E$"@pour relancer CARGEN,"
3590 PRINT"ou bien chargez le programme qui vous"
3600 PRINT"interesse apres avoir fait 'NEW'."
3610 PRINT:PRINT:PRINTEA$"Nom du jeu ";E$"W";:INPUTNJ$
3650 CLOADNJ$,A#B500,E#BB7F
6998 REM
7000 REM   BOUCLE PRINCIPALE
7002 REM
7010 AD=#BBD0:CH=#B500:XJ=3:YJ=0
7020 GOSUB7900
7100 GETX$:AS=ASC(X$)
7110 IFAS=49ORAS=50THEN7800
7150 IFAS=8THENGOTO7500
7160 IFAS=9THENGOTO7550
7170 IFAS=10THENGOTO7600
7180 IFAS=11THENGOTO7650
7200 IFAS=127THENFORM=0TO280STEP40:FORN=0TO5:POKE48687+N+M,32:NEXTN
:NEXTM
```

```
7210 IFAS=27THENRETURN
7220 IFAS=44THENGOSUB7700
7230 IFAS=46THENGOSUB7750
7240 IFAS=67ORAS=13THENGOSUB2000
7245 IFAS=32THENGOSUB7400
7250 GOTO7100
7398 REM
7400 REM   On/Off point matrice
7402 REM
7410 LA=48687+YM+XM:PRINTCHR$(17);
7420 IFPEEK(LA)=127THENPOKELA,32ELSEPOKELA,127
7430 PRINTCHR$(17);:RETURN
7498 REM
7500 REM   Mvts curseur matrice
7502 REM
7510 IFXM>0THENXM=XM-1:PRINTX$;
7520 GOTO7100
7550 IFXM<5THENXM=XM+1:PRINTX$;
7560 GOTO7100
7600 IFYM<280THENYM=YM+40:PRINTX$;
7640 GOTO7100
7650 IFYM>0THENYM=YM-40:PRINTX$;
7690 GOTO7100
7699 REM
7700 REM   Deplct dans jeu
7702 REM
7710 IFXJ>3THENXJ=XJ-2:GOTO7730ELSEPOKEAD+XJ+YJ,23:XJ=37
7720 IFYJ>0THENYJ=YJ-40ELSEYJ=JJ(J,2):XJ=JJ(J,1)
7730 GOSUB7900:RETURN
7750 POKEAD+XJ+YJ,23
7755 IFXJ<37ANDXJ+YJ<JJ(J,1)+JJ(J,2)THENXJ=XJ+2:GOTO7770ELSEXJ=3
7760 IFYJ<JJ(J,2)THENYJ=YJ+40ELSEYJ=0
7770 GOSUB7900:RETURN
7798 REM
7800 REM   Chgt de jeu
7802 REM
7810 IFVAL(X$)<>JTHENJ=VAL(X$)ELSEGOTO7100
7820 FORN=48042TO48282STEP40
7830 POKEN,A(J)
7840 NEXTN
7845 IFXJ+YJ<=JJ(J,1)+JJ(J,2)THEN7850
7847 POKEAD+XJ+YJ,23:XJ=JJ(J,1):YJ=JJ(J,2):GOSUB7900
7850 GOSUB1000
7860 GOTO7100
7900 REM
7920 POKEAD+XJ+YJ+2,23
7930 POKEAD+XJ+YJ,22
7940 RETURN
7998 REM
8000 REM   Impression du jeu
8002 REM
8010 PRINTCHR$(30):PRINT" ";
8020 FORN=0TO72STEP18
8030 FORI=32+NTO32+17+N
```

```
8040 PRINT" "CHR$(I);
8050 NEXTI
8060 PRINT"  ";
8070 NEXTN
8080 PRINT" z { | }"
8090 RETURN
8199 REM
8200 REM   Dessin Matrice
8202 REM
8204 POKE616,9:PRINT:PRINT"  "EA$:PRINT:PRINT
8210 G$=CHR$(126):G1$="    "
8215 GOSUB1000
8220 FORI=1TO8:G1$=G1$+G$:NEXT
8225 PRINTE$"G"E$"QMATRICE"E$"W"SPC(1)E$"Q":PRINTSPC(14)EA$
8226 PRINT"     ABCDEF"SPC(3)EA$:PRINTG1$SPC(2)EA$
8230 FORI=1TO8:PRINT"   "STR$(I)G$"      "G$SPC(2)EA$
8240 NEXTI:PRINTG1$SPC(2)EA$;
8260 RETURN
8498 REM
8500 REM   Aides
8502 REM
8505 KK=17
8510 PLOTKK,12,"        AIDE"
8520 PLOTKK,15,"FLECHES: Mouv. Matrice"
8530 PLOTKK,14," < >   : Mouv.Jeu Car."
8540 PLOTKK,16,"BARRE  : On/Off point"
8550 PLOTKK,19,"RETURN : Matrice->Jeu"
8560 PLOTKK,18,"C      : Jeu->Matrice"
8570 PLOTKK,20,"DEL    : Effac.Matrice"
8575 PLOTKK,22,"1-2    : Choix du Jeu"
8580 PLOTKK,23,"ESC    : SAVE/LOAD/FIN"
8590 POKE616,16:PRINT:POKE617,7:PRINTCHR$(17);:RETURN
8600 RETURN
9000 REM   En-tete
9002 REM
9020 DOKE621,47960:PRINTCHR$(30)EN$:DOKE621,48000
9030 RETURN
9098 REM
9100 REM   INITIALISATIONS
9102 REM
9110 FORI=1TO2:READA(I):READJJ(I,1):READJJ(I,2):NEXTI
9120 READ NJ$,J
9140 E$=" "+CHR$(27):EA$=E$+"U"
9160 IFPEEK(47096)=63THENFORN=0TO7ELSEGOTO9190
9170 POKE47096+N,PEEK(47088+N):POKE47088+N,0
9180 NEXT:POKE47091,8
9190 RETURN
9200 DATA 8,9,200,9,17,160
9210 DATA STANDARD,1
10000 N=609:CLS
10005 GETX$:PRINTX$;
10010 PLOT0,0,STR$(PEEK(N))+"  ":PLOT10,0,STR$(N)
10020 IFX$=CHR$(27)THENN=N+1
10030 GOTO10005
```

Structure et explications du programme

"CARGEN" est construit de façon modulaire, cinq sous-programmes servent à sa mise en route et à la constitution de l'écran, et huit autres gèrent la totalité des fonctions disponibles; l'ensemble occupe environ 4,3 Koctets en mémoire. Dans les explications, nous respecterons l'ordre d'exécution du programme plutôt que celui des numéros de ligne.

— 30: Sauvegarde des pointeurs de fin de programme, au cas où un jeu serait chargé par CLOAD (voir § 2.4.2).

— 60: Appel du sous prog. 9100 où sont effectuées les initialisations de variables et la création des deux caractères spéciaux utilisés par le programme.
Le tableau A( ) contient des codes à afficher pour le changement de jeu, et le tableau JJ( ) les pointeurs servant à l'affichage de ces jeux. E$ est le caractère ESCAPE ou CHR$(27).

— * 90-200: Appel des routines de démarrage et de constitution de l'écran:

— 9000: Gère l'en-tête affiché en première ligne d'écran.

— 8000: Ces trois sous-programmes affichent à l'écran tout ce qui est nécessaire au programme. Le sous-prog. 8000 imprimant le jeu n'est appelé qu'une fois, le changement de jeu s'obtenant par l'impression des codes adéquats dans la troisième colonne de l'écran.

— * 7000-7250: Boucle principale du programme; le clavier est lu, et en fonction de la touche appuyée le branchement est effectué vers les routines suivantes:

— 7400: Trace ou efface un point de la matrice agrandie, en fonction de l'état précédent de ce point.

— 7500: Gère les déplacements du curseur à l'intérieur de la matrice à l'aide des quatre flèches; YM est l'ordonnée et XM l'abscisse du curseur à l'intérieur de la grille.

— 7700: Gère les déplacements du curseur spécial (qui est en fait une zone colorée) dans le jeu de caractères; XJ et YJ sont l'abscisse et l'ordonnée de ce curseur.

— 7800: Opère le changement de jeu, par simple modification des codes placés au début de chaque ligne où celui-ci est affiché. NJ contient le N° du jeu (1 ou 2), et le tableau A( ) le code à imprimer. Le tableau JJ( )

contient les limites d'affichage (n'oublions pas qu'il n'y a que 64 mosaïques graphiques).

— 2000: Calcul de l'adresse de début (AA) du caractère pointé par le "curseur jeu", en fonction de la position de celui-ci (PO, elle-même calculée en fonction de XJ et YJ) et du jeu affiché (J=1 ou 2).

— 2100: Lecture des octets constituant le caractère pointé par le "curseur jeu" dans les adresses correspondantes, puis conversion en binaire afin de dessiner chaque ligne de la matrice. AW est l'adresse de début du caractère.

— 2200: Effectue l'opération inverse: Lit chaque ligne de la matrice, la convertit en décimal, puis place chaque nombre aux adresses du caractère pointé par le "curseur jeu".

— 3000: Fonctions annexes, Sauvegarde et Chargement d'un jeu de caractères sur cassette.

Note: De nombreuses REM ont été incluses dans le programme, dans un souci de clarté. Vous pouvez évidemment supprimer purement et simplement les lignes du type 1098, 2002, etc. pour vous éviter du travail de frappe, mais conservez les Numéros de ligne "ronds": 2000, 3000, 8000... car de nombreux GOTO et GOSUB y font référence.

3.8.2. Le programme "GENETEX"

Programme de traitement de textes articulé autour d'un menu principal proposant huit options différentes, Genetex est également conçu de façon modulaire: Chaque option du menu constitue pratiquement un programme indépendant exécutant une tâche précise. Son occupation-mémoire est de 9 Koctets environ. Le Menu principal vous donne le choix entre:

Ecrire un Nouveau texte

Demande la longueur de ligne désirée (jusqu'à 65 colonnes), et passe en mode "saisie"; vous pouvez taper du texte "au kilomètre", c'est-à-dire sans vous soucier des passages à la ligne. Les mots trop longs ne sont pas coupés en fin de ligne, et le texte est mémorisé dans un tableau de chaînes au fur et à mesure de son entrée; celui-ci peut faire jusqu'à 600 lignes de long.

Ajouter du texte

Imprime en haut de l'écran les cinq dernières lignes du texte en mémoire, et passe en mode "saisie".

Visualiser le texte

Permet de parcourir celui-ci rapidement en affichant les numéros de ligne.

Modifier le texte

De loin l'option offrant le plus de possibilités, elle autorise :

— L'insertion d'un nombre quelconque de lignes blanches.
— La réécriture totale ou partielle d'un ligne.
— La destruction totale ou partielle d'un ligne.
— Le réalignement d'un paragraphe sur les marges choisies après sa modification.

Imprimer le texte

Procède à l'impression sur papier du texte en mémoire, en gérant le centrage, l'interligne, le nombre de copies, les sauts de page, etc...

Sauvegarde et relecture du texte sur K7

Il serait évidemment gênant de perdre un texte sitôt l'ordinateur éteint ; ces deux options autorisent la sauvegarde du tableau sur cassette ainsi que son rechargement par la suite.

Chargement d'un jeu de caractères

Cette option opèrera le chargement d'un jeu de caractères personnel élaboré par CARGEN, par exemple un AZERTY accentué pour une présentation améliorée.

REMARQUE :

La routine de sauvegarde/lecture du tableau de chaînes sur cassette située entre les lignes 10000 et 10310 est absolument identique à celle présentée au paragraphe 2.4.2. Si vous l'avez déjà tapée, même avec des numéros de ligne différents, chargez-la *avant* de commencer à écrire le programme, cela vous évitera d'avoir à la retaper. Il suffit ensuite de modifier la ligne 110 qui appelle la routine en y mettant le numéro de ligne correct.

En ce qui concerne les REM, ce que nous avons dit à propos de CARGEN reste entièrement valable.

```
1 REM     ** GENETEX **
10 CLS
20 REM    Gestion pointeurs
22 REM
30 IFDEEK(#9C)>2E4THENDOKE#9C,DEEK(#400)ELSEDOKE#400,DEEK(#9C)
40 GRAB:HIMEM46000
100 GOSUB9000
110 GOSUB10000
120 GOTO8000
997 REM
998 REM ** SOUS-PROGRAMMES **
999 REM
1000 REM    NOUVEAU TEXTE
1002 REM
1004 REM
1006 REM    Saisie variables
1008 REM
1010 M$="NOUVEAU TEXTE":GOSUB8800
1020 CLS:POKE616,4:PRINT
1030 PRINT"ATTENTION: Si un texte se trouve en"
1040 PRINT"memoire, il sera efface.":PRINT
1050 PRINT"<RETURN>= suite, <ESC>= retour au menu":PRINT
1060 GETX$
1070 IFX$=E1$THEN RETURN
1080 IFX$=CHR$(13)THEN1100
1090 GOTO1060
1100 PRINT"Longueur des lignes, en nombre de"
1110 INPUT"caracteres (Max.=65)";LL
1130 IFLL>65ORLL<10THEN1110
1140 IFLL>39THENP1=2ELSEP1=1
1150 GOSUB7000:CLS
1250 T$="":C=1:L=1:LS=LL-4:P2=1
1260 GOSUB9900
1270 POKE#20C,127
1298 REM
1300 REM    Boucle de saisie
1302 REM
1310 C$=KEY$:IFC$=""THEN1310
1320 AX=ASC(C$)
1330 IFAX=13THEN1600
1340 IFAX=127ANDC>1THENPRINTC$;:C=C-1:T$=LEFT$(T$,C-1):GOTO1300
1350 IFAX=27 THEN RETURN
1360 IFAX=20THENPRINTC$;
1390 IFAX<32THEN1310
```

```
1398 REM
1400 REM   Impression car.
1402 REM
1410 IFC=LSTHENPING
1420 IFC=LLTHEN1500
1430 PRINTC$;
1440 T$=T$+C$:C=C+1
1450 GOTO1300
1498 REM
1500 REM   Pass. a la ligne auto
1502 REM
1505 IFAX=32THENC$=CHR$(13):GOTO1640
1510 C1=C-1:PRINTCHR$(8);
1520 IFMID$(T$,C1,1)=" "THEN1550
1530 C1=C1-1:PRINTCHR$(8);
1540 GOTO1520
1550 PRINTCHR$(9)LEFT$(ES$,C-C1);
1560 T$(L)=LEFT$(T$,C1):L=L+1:T$=RIGHT$(T$,C-C1-1):PRINTCHR$(13)
1570 PRINTT$;:C=LEN(T$)+1
1580 GOTO1410
1598 REM
1600 REM   Pass. a la ligne manuel
1602 REM
1610 T$=T$+PA$:PRINTPA$;
1640 T$(L)=T$:L=L+1:T$="":C=1:PRINTC$
1650 GOTO1300
1898 REM
1900 REM   AJOUTER DU TEXTE
1902 REM
1910 GOSUB7000:CLS
1930 IFL=0THENPOP:GOTO8000
1940 IFL>10THENL1=L-10ELSEL1=1
1950 FORN=L1TOL-1
1960 PRINTT$(N)
1970 NEXT
1980 PRINTT$(L);
1990 RETURN
1999 REM
2000 REM   MODIFICATIONS
2002 REM
2010 M$="MODIFICATIONS":GOSUB8800:CLS
2020 GOSUB8700
2030 GOSUB7200
2050 GOSUB2900
2060 T$="":C=1
2070 POKE#20C,127
2097 REM
2098 REM   Boucle principale modif
2099 REM
2100 PLOT1,18,"LIGNE No"+STR$(DL)+"  "
2110 X$=KEY$:IFX$=""THEN2110ELSEAX=ASC(X$)
2120 IFAX=12THEN2300    'Insert ligne
```

```
2130 IFAX=4 THEN2400     'Destr. ligne
2140 IFAX=18THEN2500     'Restr. parag
2150 IFX$=E1$THENRETURN:'Retour Menu
2155 IFAX=20THENPRINTX$;
2157 IFAX=5THENT$(DL)=LEFT$(T$(DL),C+(C>1)):GOTO2050
2160 IFAX=10ANDDL<LTHENDM=DL:DL=DL+1:GOTO2050
2165 IFAX=9ANDC<LLTHEN2250
2170 IFAX=11ANDDL>1THENDM=DL:DL=DL-1:GOTO2050
2180 IFAX=127ANDC>1THENPRINTX$;:C=C-1:T$=LEFT$(T$,C-1):GOTO2110
2190 IFAX=13THENT$(DL)=T$+PA$:C=1:GOTO2050
2195 IFAX<32THEN2110
2198 REM
2200 REM   Impression caractere
2202 REM
2210 IFC=LLTHEN2110
2220 PRINTX$;:T$=T$+X$:C=C+1
2230 GOTO2110
2248 REM
2250 REM   Deplct Horiz. Curseur
2252 REM
2260 T$=T$+MID$(T$(DL),C,1)
2270 PRINTX$;:IFC=38THENPOKE617,2
2280 C=C+1:GOTO2110
2298 REM
2300 REM   Insert lignes
2302 REM
2310 CLS:PRINT:PRINT:PRINT
2320 INPUT"Combien de lignes a inserer ";IN
2330 IFIN+L>40000/LL+3THENPRINT:PRINT"  PLUS DE PLACE !":PRINT:GOTO
2320
2340 FORN=(L+IN)TO(DL+IN)STEP-1
2350 T$(N)=T$(N-IN)
2360 NEXT
2370 FORN=DLTODL+IN-1
2380 T$(N)=PA$
2390 NEXT:L=L+IN
2395 C=1:GOTO2050
2398 REM
2400 REM   Destr  lignes
2402 REM
2410 FORN=DLTOL-1
2420 T$(N)=T$(N+1)
2430 NEXT:L=L-1
2440 C=1:GOTO2050
2498 REM
2500 REM   Refonte texte
2502 REM
2510 U=0:CLS:PRINT:PRINT:PRINTE$"L         REFONTE EN COURS"
2520 M=DL
2530 N=LEN(T$(M))+2:A=0:T1=LEN(T$(M+1))
2540 REPEAT
2550 N=N-1
```

```
2560 UNTILMID$(T$(M),N,1)<>" "ORN=1
2570 PL=(LL-N)
2580 C=0
2590 REPEAT
2600 C=C+1:IF MID$(T$(M+1),C,1)=" "THENA=C
2605 IF MID$(T$(M+1),C,1)=PA$THENA=C+1
2610 UNTILC>PL
2620 IFA=0THEN2650
2630 T$(M)=LEFT$(T$(M),N)+" "+LEFT$(T$(M+1),A-1)
2635 IFA>T1THENA=T1
2640 T$(M+1)=RIGHT$(T$(M+1),T1-A)
2650 M=M+1
2660 IFRIGHT$(T$(M-1),1)=PA$THENC=1:T$(M)=PA$:GOTO2680
2665 IF RIGHT$(T$(M),1)=PA$THENC=1:GOTO2680
2670 GOTO2530
2680 IFUTHEN2050ELSEU=-1:GOTO2520
2698 REM
2900 REM   Affichage lignes
2901 REM
2902 IFC=1THEN2910
2904 IFC<=LEN(T$(DM))THENTD$=RIGHT$(T$(DM),(LEN(T$(DM))-C)+1)ELSETD
$=""
2906 T$(DM)=T$+TD$
2910 CLS:PRINT:IFDL-8/P1<1THENMI=0ELSEMI=DL-6/P1
2915 IF(DL+4/P1)<LTHENMA=(DL+6/P1)ELSEMA=L
2917 POKE616,1:PRINT
2920 FORN=MITODL-1:PRINTT$(N):NEXT
2930 PO=PEEK(616):PRINTT$(DL)
2940 FORN=DL+1TOMA:PRINTT$(N):NEXT
2950 POKE616,PO-1:PRINT
2960 POKE617,2
2990 DM=DL:RETURN
2998 REM
3000 REM   IMPRESSION
3002 REM
3004 REM   Saisie variables
3006 REM
3010 M$="IMPRESSION"
3020 GOSUB8800:CLS
3100 POKE616,5:PRINT
3110 PRINT"Marge (Max="67-LL") Centre=100 ";
3120 INPUT MG
3124 IFMG=100THENMG=(78-LL)/2
3126 IFMG>67-LLTHENMG=67-LL
3130 PRINT:INPUT"Simple/double interligne (1/2) ";IT
3140 PRINT:INPUT"Ligne debut ";L1
3150 PRINT:INPUT"Ligne fin   ";L2
3155 IFL2>LTHENL2=L
3160 PRINT:INPUT"Lignes/page ";PP
3170 PRINT:INPUT"Nombre de copies ";NC
3197 REM
3198 REM   Boucle d'impression
3199 REM
```

```
3200 GOSUB7300:CLS
3210 :    FORM=1TO NC
3220 :    GOSUB 3350
3230 :    N=L1
3240 : REPEAT
3250 : IFRIGHT$(T$(N),1)=PA$THENP$=LEFT$(T$(N),LEN(T$(N))-1)ELSEP$=
T$(N)
3260 : PRINTP$:LPRINTSPC(MG)P$
3270 : IFIT=2THENLPRINT
3280 : IFN/PP/IT=INT(N/PP/IT)THENLPRINTCHR$(12);:GOSUB3350: REM Chg
t. Page
3290 : N=N+1
3300 : UNTIL N>L2 OR X$=E1$
3310 :    LPRINTCHR$(12);
3320 :    NEXT M
3330 :    RETURN
3340 REM
3350 : FORI=1TO(65-PP)/2:LPRINT:NEXT:RETURN
3998 REM
4000 REM   VISUALISATION
4002 REM
4010 M$="VISUALISATION":GOSUB8800:CLS
4020 GOSUB 8700
4100 GOSUB7100
4105 CLS:PRINT
4110 IFDL<1ORDL>LTHENDL=1
4120 N=DL
4130 REPEAT
4140 PRINTT$(N)
4150 N=N+1
4170 UNTILN=DL+20/P1ORN=L+1
4190 PLOT2,22,"LIGNES "+STR$(DL)+"-"+STR$(N)+"   "
4200 GETX$
4210 IFX$=CHR$(10)THENDL=DL+16/P1:GOTO4105
4220 IFX$=CHR$(11)THENDL=DL-16/P1:GOTO4105
4230 IFX$=CHR$(27)THENRETURN
4240 GOTO4190
4998 REM
5000 REM   SAVE/LOAD
5002 REM
5010 REM   Save
5012 REM
5020 CLS:IFX$="L"THEN5200
5025 IFL=0THEN POP:GOTO8000
5030 M$="SAUVEGARDE EN COURS":GOSUB8800:POKE48012,0:POKE616,9:PRINT
5040 T$(0)=RIGHT$(STR$(L),LEN(STR$(L))-1)+RIGHT$(STR$(LL),2)+RIGHT$
(STR$(P1),1)
5060 PRINTCHR$(4)E$"J METTEZ LE K7 EN ENREGISTREMENT":PRINTCHR$(4):
PRINT
5070 PRINT"         ET PRESSEZ UNE TOUCHE"
5080 GETX$:CLS:POKE48012,12
5090 POKE#67,0:CALL#B9A2,T$
```

```
5100 RETURN
5198 REM
5200 REM    Load
5202 REM
5210 M$="CHARGEMENT EN COURS":GOSUB8800:POKE48012,0:POKE616,9:PRINT
5220 PRINTCHR$(4)E$"J      DEMARREZ LA CASSETTE":PRINTCHR$(4):PRINT
5230 PRINT"        ET PRESSEZ UNE TOUCHE"
5240 GETX$:CLS:POKE48012,12
5250 GOSUB9900
5260 POKE#67,0:CALL#B9A5,T$
5270 N=LEN(T$(0))-3
5280 L=VAL(LEFT$(T$(0),N)):LL=VAL(MID$(T$(0),N+1,2)):P1=VAL(RIGHT$(
T$(0),1))
5290 LS=LL-4:T$="":C=1:P2=1:T$(0)=""
5300 RETURN
6998 REM
7000 REM   7000-7400=TEXTES D'AIDE
7002 REM   ----------------------
7010 REM   Saisie
7012 REM
7020 POKE616,21:PRINT:POKE623,20
7030 PRINTE$"V<MARGE"TAB(LL+3)">"E1$"W"
7040 IFLL>39THENPRINTCHR$(11)E1$"V"
7050 POKE616,23:PRINT:PRINT"-RETURN  fin de paragraphe"
7060 PRINT"-DEL      efface le caractere precedent";
7070 PRINT"-ESCAPE  retour menu"
7080 RETURN
7098 REM
7100 REM   Visu
7102 REM
7110 POKE616,22:PRINT:PRINTE1$"V":PRINT
7130 PRINT"<^> ou <"CHR$(126)">   Deplacement"
7140 PRINT"<ESC>     Retour Menu"
7150 POKE623,21:RETURN
7198 REM
7200 REM   Modif
7202 REM
7210 POKE616,18:PRINT:PRINTE1$"V":PRINT
7220 PRINT"<^><"CHR$(126)">   Balayage ecran"
7230 PRINT"-> <DEL> Balayage ligne"
7240 PRINT"<CTRL E> Effacement ->Fin de ligne"
7250 PRINT"<CTRL L> Insertion lignes vides"
7260 PRINT"<CTRL D> Destruction ligne"
7270 PRINT"<CTRL R> Restructuration paragraphe"
7280 PRINT"<ESC>    Retour Menu";
7290 POKE623,18:RETURN
7298 REM
7300 REM   Impression
7302 REM
7310 POKE616,24:PRINT:PRINTE1$"V"E1$"G      ALLUMEZ L'IMPRIMANTE":P
RINT
7320 PRINT"<ESC>    Retour Menu"
7330 POKE623,23:RETURN
7997 REM
```

```
7998 REM --------------------
7999 REM !                  !
8000 REM ! MENU   PRINCIPAL !
8002 REM !                  !
8004 REM --------------------
8006 REM
8010 M$=M1$:GOSUB8800
8020 POKE#20C,255:POKE#26F,27
8030 CLS:POKE#268,5
8100 PRINTCHR$(4):PRINTE$"J    MENU PRINCIPAL"
8110 PRINTCHR$(4):PRINT:PRINT
8120 PRINT"   <N>ouveau texte"
8130 PRINT"   <A>jouter au texte en memoire"
8140 PRINT"   <V>isualiser le texte en memoire"
8150 PRINT"   <M>odifier le texte en memoire"
8160 PRINT"   <I>mprimer le texte en memoire":PRINT
8170 PRINT"   <S>auvegarder le texte sur K7"
8180 PRINT"   <L>ire un texte sur K7"
8190 PRINT"   <C>harger un jeu de caracteres"
8200 PRINT:PRINT"    VOTRE CHOIX ?";:GETX$
8210 IFX$="N" THEN GOSUB1000
8220 IFX$="C"THEN8500
8230 IFX$="L"ORX$="S"THENGOSUB5000
8240 IFL=0THEN8000
8260 IFX$="M" THEN GOSUB2000
8270 IFX$="I" THEN GOSUB3000
8280 IFX$="V" THEN GOSUB4000
8290 IFX$="A" THEN GOSUB1900:GOSUB1270
8300 GOTO8000
8498 REM
8500 REM   Load jeu de CHR
8502 REM
8510 M$="Load Jeu":GOSUB 8800:CLS
8520 PRINT:PRINT:PRINT
8530 PRINT"Quand le jeu est charge, tapez GOTO 1"
8540 PRINT"pour relancer le programme.":PRINT:PRINT
8550 INPUT"NOM DU JEU DE CHR : ";NJ$
8560 CLOADNJ$,A#B500,E#BB7F
8700 REM
8710 POKE616,5:PRINT
8720 PRINT"Ligne de debut (MAX="L") ";:INPUT DL
8730 CLS:RETURN
8798 REM
8800 REM   En-tete
8802 REM
8810 DOKE621,47960:PRINTCHR$(30)CHR$(14)M$:DOKE621,48000:DOKE48000,
1808
8820 RETURN
8898 REM
9000 REM   Initialisations
9002 REM
9010 M1$="GENETEX - TRAITEMENT DE TEXTES  "+CHR$(96)+" BVRP":M$=M1$
:GOSUB 8800
```

```
9020 PLOT8,10,"UN PEU DE PATIENCE !"
9030 ES$="                                        "
9040 E1$=CHR$(27):E$=" "+E1$:PA$=CHR$(126)
9050 REM
9300 READAD
9310 IFAD=0THEN9800
9320 FORN=0TO7
9330 READDA:POKEAD+N,DA
9340 NEXT
9350 GOTO9300
9700 DATA 47088,8,8,8,8,42,28,8,0
9710 DATA 0
9800 RETURN
9900 REM   Dim tableaux
9902 REM
9910 DOKE#A0,DEEK(#9E)
9920 DIMT$(600)
9930 DIMI$(15)
9960 RETURN
10000 REM  Save/Load tableau
10010 ST=#B800
10020 READ DA:REPEAT
10030 POKEST,DA:ST=ST+1
10040 READDA:UNTILDA=33
10050 DOKE#B9A2,#0A4C:DOKE#B9A4,#4CB8:DOKE#B9A6,#B858
10060 RETURN
10100 DATA85,85,85,85,35,57,68,54,56,85,32,11,185,8,32,214,184,32,1
86,230,169
10110 DATA37,32,198,229,165,51,32,198,229,165,52,32,198,229,32,238,
184,32,167
10120 DATA229,36,40,16,3,32,53,184,32,4,232,40,96,160,0,177,1,240,2
3,170,160,2
10130 DATA177,1,153,208,0,136,208,248,232,202,240,8,177,209,32,198,
229,200,208
10140 DATA245,32,195,184,144,222,96,32,149,213,32,11,185,8,32,214,1
84,32,150
10150 DATA230,32,48,230,201,37,208,249,32,48,230,133,51,32,48,230,1
33,52,160
10160 DATA2,177,206,197,51,200,177,206,229,52,176,6,32,4,232,76,131
,196,32,238
10170 DATA184,32,235,228,36,40,16,3,32,155,184,32,4,232,40,96,160,0
,177,1,240
10180 DATA28,32,240,212,170,232,160,0,202,240,8,32,48,230,145,209,2
00,208,245
10190 DATA160,2,185,208,0,145,1,136,208,248,32,195,184,144,217,96,2
4,169,3,101
10200 DATA1,133,1,168,144,2,230,2,165,2,196,97,229,98,96,32,202,230
,32,24,185
10210 DATA160,3,177,206,170,136,177,206,233,1,176,1,202,133,51,134,
52,96,24,165
10220 DATA206,101,51,133,97,165,207,101,52,133,98,160,4,177,206,32,
246,209,133
```

```
10230 DATA95,132,96,133,1,132,2,96,32,232,0,201,44,240,3,76,228,207
,76,226,0
10240 DATA162,0,32,232,0,134,39,133,180,32,232,0,32,134,209,176,6,3
2,4,232,76
10250 DATA228,207,162,0,134,40,134,41,32,226,0,144,5,32,134,209,144
,11,170,32
10260 DATA226,0,144,251,32,134,209,176,246,201,36,208,6,169,255,133
,40,208,12
10270 DATA201,37,208,15,169,128,133,41,5,180,133,180,138,9,128,170,
32,226,0
10280 DATA134,181,166,158,165,159,134,206,133,207,197,161,208,4,228
,160,240,31
10290 DATA160,0,177,206,200,197,180,208,6,165,181,209,206,240,14,20
0,177,206
10300 DATA24,101,206,170,200,177,206,101,207,144,215,56,96,32,4,232
,162,42,76
10310 DATA133,196,85,33
```

Fonctionnement du programme

— 30: Mémorisation des pointeurs de fin de programme (pour le cas du chargement d'un jeu de caractères).

— 100-120: appel des sous-programmes de mise en route.

Mise en route

— 9000: Initialisations: M1$ est le message placé en en-tête sur la ligne réservée. Création du caractère "flèche vers le bas", qui indique les fins de paragraphe et se trouve au code 126 (PA$).

— 8800: Impression sur la ligne réservée du message contenu dans M$.

— 10000: Chargement à la place du jeu de caractères graphiques de la routine en langage machine gérant la sauvegarde et relecture des textes sur cassette.

— 8000-8300: Menu principal. Celui-ci effectue l'aiguillage vers les divers sous-programmes, et il faut y retourner pour passer d'une fonction à une autre. Le clavier est mis automatiquement en majuscules pour la saisie des commandes.

Nouveau texte

— 1000-1150: Nouveau texte; saisie et initialisations de ses pointeurs et variables:

LL : Longueur des lignes.

T$: Ligne en cours de frappe.
T$( ) : Tableau, contenant l'ensemble des lignes du texte.
C : Numéro de colonne du caractère en cours de frappe à l'intérieur de la ligne.
L : Longueur du texte déjà tapé, ou N° de la dernière ligne.
LS : N° de colonne où intervient la sonnerie.
P1 : Pointeur déterminant si une ligne de texte est plus ou moins longue qu'une ligne d'écran (LL>38 ou LL<38). Utilisé par les sous-programmes "Ajouter du texte" et "Modifications".

— 1270 : Passage automatique du clavier en minuscules pour la frappe du texte.

— 1300-1390 : Boucle principale en mode "saisie". C$ est le caractère tapé.

— 1400-1450 : Gère l'impression du caractère, l'ajoute à T$, incrémente C et actionne la sonnerie trois colonnes avant la fin de ligne.

— 1500-1580 : si le dernier caractère de la ligne n'est pas un espace le mot est considéré comme trop long ; il est relu en arrière jusqu'à trouver un espace (position C1), puis soustrait de T$. La ligne obtenue est chargée dans T$(L), et le bout de mot extrait dans T$. Le curseur passe à la ligne suivante, où le nouveau T$ est imprimé. Enfin, le curseur se place à la fin de celui-ci, position d'affichage du prochain caractère.

— 1600-1650 : Passage manuel à la ligne suivante (fin de paragraphe), et impression du symbole de fin de paragraphe.

Ajout de texte

— 1900-1990 : Affiche les dix dernières lignes du texte, et va à la section "saisie", comme pour un nouveau texte (1270—) ; Le branchement s'effectue à partir du menu (8290).

Modifications

— 2020-2030 : Impression de l'en-tête et du texte d'aide.

— 2050 : Saisie du numéro de la ligne à corriger (8700), puis appel de la routine d'impression de celle-ci avec les cinq précédentes et suivantes (2900-2990). Le curseur peut ensuite être déplacé dans le texte et *dans les quatre directions avec les flèches.*

DL : Numéro de la ligne où se trouve le curseur.

P1,MA,MI : Pointeurs gérant l'impression des lignes plus longues qu'une ligne d'écran.
PO : Position verticale du curseur à l'écran.
DM : Mise en mémoire du N° de la ligne en cours, pour les déplacements verticaux.

— 2100-2195 : Boucle principale de saisie du texte modifié et des caractères de contrôle des fonctions spéciales.

— 2200-2230 : Saisie-impression d'un caractère.

— 2250-2280 : Gestion des déplacements du curseur dans la ligne.

— 2300-2395 : Insertion de IN lignes vides.

— 2400-2440 : Destruction de la ligne où se trouve le curseur.

— 2500-2680 : Si des mots ont été supprimés sur des lignes, reformate le paragraphe de manière à ce que chaque ligne contienne le maximum de mots possible.

— 2900-2990 : Ce sous-programme réaffiche l'écran chaque fois qu'on change de ligne, permettant de tenir compte des modifications survenues.

Impression

— 3000-3170 : Saisie des paramètres d'impression.

MG : Marge gauche.
IT : Simple ou double interligne (1 ou 2).
L1 : Première ligne de la portion de texte à imprimer.
L2 : Dernière ligne à imprimer (Si L2>longueur du texte, celui-ci est imprimé jusqu'à la fin).
PP : Nombre de lignes par page.
NC : Nombre de copies à effectuer.

— 3200-3330 : Boucle d'impression.

Note : Cette routine fonctionne avec tous les types d'imprimantes.
P$: Ligne à imprimer.
N : Numéro de la ligne à imprimer dans T$( ).

— 3250 : Suppression du caractère PA$ (fin de paragraphe-flèche vers le bas) avant impression.

— 3280 : Commande le changement de page lorsque le nombre correct de lignes y a été imprimé. CHR$(12) est le code de saut de page sur la plupart des imprimantes.

— 3350: Routine imprimant des lignes blanches en haut de chaque page, pour y centrer le texte en fonction de PP. Si vous utilisez du papier de 12 pouces, changez le (65-PP) en (70-PP).

Visualisation du texte en mémoire

— 4000-4240: Affiche le texte par blocs de 20 lignes (ou dix lignes si elles sont plus longues que 38 car.), avec leurs numéros.

Sauvegarde, Chargement d'un texte

— 5040: Charge T$(0) avec les paramètres du texte, sa longueur (L), la longueur des lignes (LL) et le pointeur P1.

— 5000-5060: Exécute la sauvegarde du tableau T$ en utilisant la routine en langage-machine implantée par les lignes 10000-10310.

— 5200-5260: Procède au chargement de T$( ) en mémoire à partir du magnétocassette.

— 5280: Reconvertit en variables les paramètres constituant T$(0).

Note : Si un problème se pose au cours de la lecture, le programme sera perdu et il faudra le recharger également.

Textes d'aide

— 7000-7330: Affichage de petits textes d'aide (4-5 lignes) dans une zone réservée en bas d'écran. Il y a un texte par sous-programme.

Chargement d'un jeu de caractères

— 8500-8560: Effectue le chargement d'un jeu de caractères enregistré sur K7, en particulier les jeux élaborés par CARGEN.

— 8800: Dimensionne l'écran pour récupérer la ligne du haut, y imprime l'en-tête contenue dans M$ (différente pour chaque sous-programme), puis le remet à sa taille normale.

4

Le graphique haute-résolution

4.1. *Introduction*

En plus des modes TEXT et LORES, ne permettant de dessiner qu'avec des caractères déjà formés, ORIC dispose d'un mode graphique haute résolution de 240 par 200 points adressables individuellement. Tout l'écran est accessible, même les deux colonnes et la ligne normalement interdites en mode texte.

Les huit couleurs sont toujours utilisables, mais on ne peut pas définir séparément la couleur de chaque point : Celle-ci est obtenue par un attribut placé dans une "zone" rectangulaire, dont la dimension minimale est de un point de haut par six points de long. L'attribut choisi est actif pour tout ce qui s'affiche à droite de cette zone, rappelant ainsi les codes ESCAPE en mode texte (voir §§ 3.3.-3.4) ; les attributs sont d'ailleurs exactement les mêmes.

Nous verrons dans cette partie comment exploiter toutes les commandes du graphique, mais aussi comment créer celles qui n'existent pas.

4.2. *Organisation-mémoire du graphique*

En mode texte, nous avons vu que chaque case de l'écran occupait un octet, méthode qui serait impossible en graphique s'il en était de même pour chaque point: Il ne faudrait pas moins de 240×200=48000 octets pour stocker l'écran, c'est-à-dire la totalité de la mémoire! Une autre méthode est donc utilisée qui consiste à grouper les points horizontalement, six d'entre eux constituant un octet. De cette manière, l'écran "tient" dans 200×240/6=8K., ce qui est déjà beaucoup plus raisonnable.

Le dessin déterminé par un octet est l'image exacte de sa valeur binaire, un point "allumé" (couleur 1er plan) valant 1 et un point "éteint" (couleur fond) zéro. La valeur décimale de chaque octet est trouvée selon le principe déjà vu au paragraphe 3.7.2, utilisé pour le codage des caractères.

On sait qu'un octet est constitué de 8 bits, mais ORIC n'en utilise que 6 pour coder le graphique; les bits 5 et 6 (poids 32 et 64) de l'octet déterminent si celui-ci doit être interprété comme un dessin ou comme un attribut: Si les deux sont à 0 il s'agit d'un attribut, si l'un des deux est à un c'est un dessin. (Voir aussi les commandes FILL et PATTERN, § 4.3.1).

Adresses écran

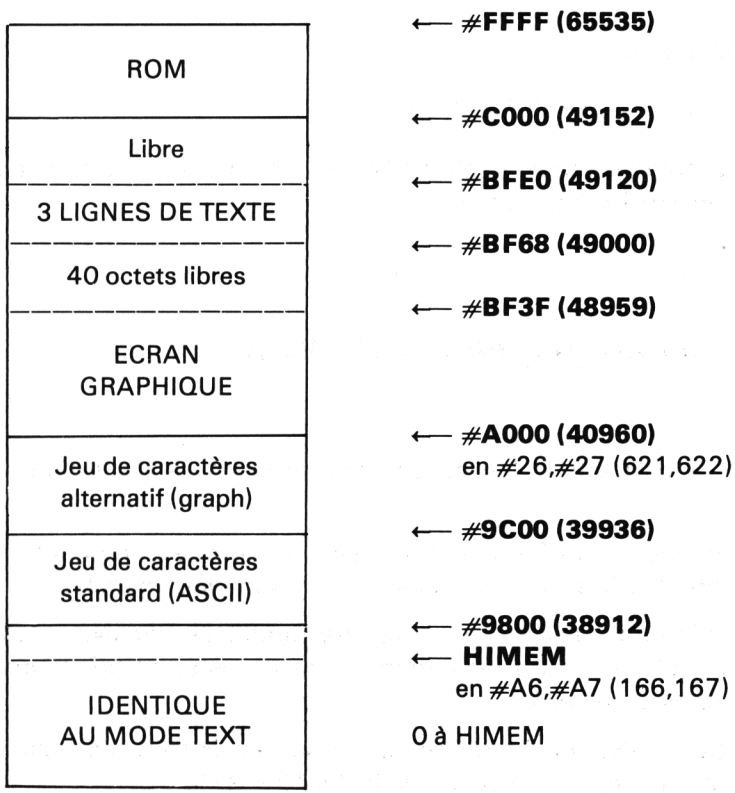
Abscisse → / Ordonnée ↓	0 1 2 3 4 5	6 7 8 9 10 11		226 – 232	233 – 239
0	40960	40961		40998	40999
199	48920	48921		48958	48959

On trouvera l'adresse-mémoire de l'octet codant un groupe de points par la formule suivante, où Y est l'ordonnée du groupe et X l'abscisse de n'importe lequel des six points :

<Adresse >=40960+Y*40+X/6

Le Memory-map en haute résolution

L'écran graphique occupant plus de place en mémoire que son homologue normal, le memory-map en est légèrement modifié :



Nous voyons que trois lignes de texte sont disponibles en bas d'écran pour des commentaires ou explications, ou pour lister des lignes de votre programme. Celles-ci ne sont pas complètement jointives avec la zone haute-résolution, une quarantaine d'octets restant libres. Leur adresse de début est #BF68 (49000), toute la zone comprise entre celle-ci et la fin de

l'écran est gérée comme en mode TEXT: Scrolling en bas d'écran, effacement par CLS etc...

En pratique, nous n'agirons pas sur les jeux de caractères en mode graphique (c'est beaucoup plus simple en TEXT), et les limites de l'écran n'ont besoin d'être connues que si on veut le sauvegarder sur cassette ou en faire une copie sur imprimante (hardcopy), méthode présentée au paragraphe 4.4.

Pour sauvegarder l'écran sur cassette, il faut être en mode HIRES puis taper:

```
CSAVE"< Nom > ",A #A000,E #BFE0 [,S]
```

Et pour le recharger:

```
CLOAD ["< Nom > "],A0,E0, [,S]
```

Enfin une variable-système, #2C0 (704), est liée au graphique: Elle prend la valeur 3 en HIRES et 2 en TEXT.

4.3. *Les commandes graphiques du Basic*

Pour utiliser à fond toutes les possibilités de la haute résolution, il faut savoir que:

— Les coordonnées de l'écran vont de gauche à droite et de haut en bas, donc le point 0,0 se trouve dans le coin supérieur gauche.

— Si les couleurs de l'écran-texte ont été modifiées par INK et PAPER *avant* d'appeler la haute résolution, celles-ci resteront effectives pour les trois lignes de texte.

— Par contre, si INK et PAPER sont utilisés en HIRES, ils agiront sur l'écran graphique et non plus sur la fenêtre-texte.

— Toutes les commandes applicables au mode texte fonctionnent sur cette fenêtre, sauf PLOT qui provoque un message d'erreur.

— Par la suite, nous utiliserons constamment la notion de "curseur graphique" que nous appellerons simplement curseur. Celui-ci détermine l'endroit de l'écran où se fera le prochain tracé, mais n'est pas

forcément visible comme le curseur texte. Quand on passe en mode HIRES, le curseur est placé aux coordonnées 0,0 (coin supérieur gauche).

— Les instructions graphiques, en dehors de leurs arguments spécifiques, utilisent toutes un paramètre définissant la couleur dans laquelle s'effectue le tracé que nous appellerons C, et qui peut prendre les valeurs suivantes :

0 : Dessine avec la couleur du fond (donc efface les points déjà tracés).

1 : Dessine dans la couleur définie pour le premier plan.

2 : Inverse la couleur des points sur lesquels passe le curseur (Agit comme 0 sur un point déjà tracé et comme 1 s'il n'y a rien).

3 : Ne fait rien (aucun point tracé).

4.3.1. Instructions

Commandes d'intérêt général

HIRES

Syntaxe : HIRES

Application : Démarre le mode haute-résolution si on est en mode texte, et efface l'écran graphique si on est déjà dans ce mode. Ce dernier n'est pas affecté par CLS ou CTRL L, qui agiront sur les trois lignes de texte.

TEXT ou LORES N

Syntaxe : TEXT ou LORES N avec N=0, N=1

Application : Repasse en mode texte.

Commandes de dessin

CURSET

Syntaxe : CURSET <Abscisse>,<Ordonnée>,<C>

Application : Place le curseur au point spécifié. Cette commande est l'équivalent de PLOT ou PSET dans d'autres Basic.

Exemple: CURSET 120,100,1 met le curseur au centre de l'écran.

CURMOV

Syntaxe: CURMOV<X>,<Y>,C

Application: Ajoute X et Y aux coordonnées courantes du curseur, et place celui-ci à la position obtenue (déplacement relatif).

DRAW

Syntaxe: DRAW <X>,<Y>,C

Application: Trace une droite entre la position du curseur et le point obtenu en y ajoutant X et Y (déplacement relatif).

REMARQUE: Si la deuxième extrémité de la droite se trouve plus haut ou à gauche de la position du curseur, X et Y peuvent bien sûr avoir une valeur négative.

CIRCLE

Syntaxe: CIRCLE <Rayon>,C

Application: Trace un cercle du rayon spécifié, centré sur la position du curseur.

REMARQUES:

— Vous devrez veiller à ce que le cercle ne dépasse pas les limites de l'écran, ce qui provoquerait une "ILLEGAL QUANTITY ERROR".

— Si vous utilisez la valeur 2 pour C, tous les points du cercle ne seront pas tracés: L'approximation de calcul ayant pour effet d'écrire deux fois de suite certains points, ceux-ci sont effacés au deuxième passage.

Exemple: Dessin d'un cercle plein (boule):

```
10 CURSET 120,100,3
20 FOR N=1 TO 50
30 CIRCLE N,1
40 NEXT N
```

CHAR

Syntaxe: CHAR <Code Ascii>,<Jeu>,C

Application: Écrit le caractère ayant le code spécifié à la position du curseur. Si <Jeu>=0, le caractère est pris dans le jeu standard et si <jeu>=1 c'est le jeu semi-graphique qui est utilisé.

REMARQUES:

— La position du curseur définit le coin supérieur gauche de la matrice du caractère.

— Après écriture le curseur demeure au même endroit, ses déplacements doivent donc être entièrement gérés par l'utilisateur.

— Le seul moyen d'effacer un caractère est de le réécrire au même endroit, en donnant à C la valeur 0 ou 2.

Exemple: Le petit programme suivant écrit le message contenu dans M$ au centre de l'écran.

```
10 HIRES:M$="ECRITURE HAUTE RESOLUTION"
20 CURSET (240-LEN(M$)*6)/2,90,3
30 FOR N=1 TO LEN(M$)
40 CHAR ASC(MID$(M$,N,1)),0,1
50 CURMOV 6,0,3
60 NEXT N
```

— La ligne 20 centre la phrase en fonction de sa longueur.

— La ligne 40 prend le code de chaque caractère extrait de M$, et le donne comme argument à CHAR.

— La ligne 50 fait avancer le curseur de 6 points (largeur d'un caractère) après chaque écriture.

Définition d'une zone

FILL

Syntaxe: FILL <hauteur>,<largeur>,<N>

Application: Remplit une zone dont le coin supérieur gauche est marqué par la position du curseur, et dont la forme peut être définie de deux manières différentes selon la valeur donnée à N:

1) Si N est compris entre 0 et 31, il est considéré comme un attribut exactement comme pour PLOT (cf.§ 3.4); les mêmes valeurs sont d'ailleurs applicables. L'attribut est alors en vigueur sur un rectangle situé à droite de la zone ainsi créée, celle-ci étant le côté gauche du rectangle.

2) Si N est compris entre 32 et 127, il est considéré comme un motif correspondant à son équivalent binaire sur six Bits. Un des deux Bits 5 ou 6, respectivement de poids 32 et 64, doit obligatoirement être à 1.

REMARQUES :

La hauteur est définie en nombre de points, mais la largeur est définie en *nombre de cellules de 6 points*. Elle pourra donc avoir une valeur comprise entre 1 et 40.

— Après exécution de FILL, le curseur est déplacé au coin inférieur gauche de la zone définie indépendamment de sa largeur, comme si on avait exécuté CURMOV 0,<hauteur>,C.

— Lorsque FILL définit un attribut, il est impossible de dessiner dans les cellules qui contiennent ce dernier.

Le tableau suivant montre la correspondance existant entre le tracé apparaissant effectivement à l'écran, et les valeurs dédimale et binaire de N :

Cellule obtenue par FILL 1,1,<N>									*Valeur décimale de N*	*Équivalent binaire*
Bits→	7	6	**5**	**4**	**3**	**2**	**1**	**0**		
			*						32	00 **100000**
			*		*		*		42	00 **101010**
			*	*	*				56	00 **111000**
			*	*	*	*	*	*	63	00 **111111**
		*							64	01 **000000**
		*						*	65	01 **000001**

Les bits marqués en **gras** sont ceux effectivement pris en compte pour l'élaboration du motif. Nous constatons que dans tous les cas, soit le Bit 5 soit le Bit 6 est à 1 : Comme il est impossible de donner à N une valeur inférieure à 32 qui le définirait comme un attribut, on met à 1 le Bit 6 qui n'entre pas dans le dessin de la cellule elle-même ; cela revient en fait à ajouter 64 à la valeur décimale obtenue par les 6 Bits de cette dernière.

Rappelons enfin que le poids d'un bit est trouvé en élevant 2 à la puissance de son N° d'ordre : Le Bit 7 vaut $2^7=128$, le Bit 5 vaut $2^5=32$, etc... (cf. § 2.1).

***Exemples*:**

Programme créant une zone de 120 points de large par 100 points de haut au centre de l'écran, où l'affichage se fait en jaune clignotant sur bleu:

```
10 HIRES
20 CURSET 60,50,3:FILL 100,1,20
30 CURSET 66,50,3:FILL 100,1,3
40 CURSET 180,50,3:FILL 100,1,7
50 CURSET 186,50,3:FILL 100,1,16
60 CURSET 120,100,3:FOR N=1 TO 30 STEP 3:CIRCLE N,1:NEXT
70 WAIT 40:CURSET 54,50,3:FILL 100,1,12
```

— Lignes 20 à 50: Les attributs de couleur sont placés aux colonnes 60 et 66, puis remis aux valeurs d'origine aux colonnes 174 et 180.

— Ligne 60, création d'un dessin.

— Ligne 70, l'attribut "clignotant" (12) est mis à la colonne 54.

Pour dessiner dans la zone précédente un motif "3 points 2 blancs un point", remplacer la ligne 60 par:

```
60 CURSET 80,66,3:FILL 68,15,57
```

Le dessin de valeur décimale 57 (111001) remplit une zone de 68 points de haut par 15*6=90 points de large.

Définition du trait

PATTERN

Syntaxe: PATTERN <N>

Application: Sélectionne le type de trait utilisé par DRAW et CIRCLE, qui est fonction de l'équivalent binaire de N sur 8 Bits. N est donc compris entre 0 et 255. La valeur à la mise sous tension ou à l'appel du mode graphique par HIRES est de 255 (trait plein).

REMARQUE: On trouve l'équivalent décimal du type de trait désiré exactement de la même façon que pour FILL, avec la différence qu'il est constitué de 8 bits au lieu de 6.

Notez aussi que N est remis à 255 chaque fois qu'un HIRES (effacement d'écran) est exécuté.

Exemple: Le programme suivant vous permet d'expérimenter tous les types de trait possibles sur le dessin d'un rectangle. L'appui sur les flèches "haut" ou "bas" augmentera ou diminuera la valeur servant d'argument à PATTERN.

```
10 HIRES:PATTERN N
20 ?:?"MOTIF NO"N
30 CURSET 60,60,3:DRAW 120,0,1:DRAW 0,80,1:DRAW-120,0,1:
   DRAW 0,-80,1
40 GET X$
50 IF X$=CHR$(10) AND N>0 THEN N=N-1
60 IF X$=CHR$(11) AND N< 255 THEN N=N+1
70 GOTO 10
```

4.3.2. Fonction

POINT

Syntaxe:

POINT(<abscisse>,<ordonnée>)

Application: seule fonction spécifique du graphique, elle retourne 1 si le point aux coordonnées fournies est dans la couleur du 1er plan, et 0 s'il est dans la couleur du fond.

4.4. Quelques programmes

Déplacer le curseur et dessiner dans huit directions

Ce programme autorise le déplacement du curseur dans les quatre directions principales avec les flèches, et dans les 4 diagonales par combinaison des flèches et de la touche SHIFT. On peut écrire ou effacer en pressant les touches "1" ou "0", et effacer l'écran avec DEL:

```
  5 REM PAPIER & CRAYON
  7 REM
 10 HIRES:HIMEM 9000
 20 X=100:Y=100:T=0
 30 ?:?"TRAIT=0"
 40 CURSET XM,YM,T:CURSET X,Y,1
 50 X$=KEY$:IF X$="" THEN 50
 60 XM=X:YM=Y
 70 IF PEEK(521)=56 THEN 130
 80 IF X$=CHR$(8) AND X>0 AND Y>0 THEN X=X-1:Y=Y-1
 90 IF X$=CHR$(10) AND X>0 AND Y<199 THEN X=X-1:Y=Y+1
100 IF X$=CHR$(11) AND X<239 AND Y>0 THEN X=X+1:Y=Y-1
110 IF X$=CHR$(9) AND X<239 AND Y<199 THEN X=X+1:Y=Y+1
120 GOTO 40
130 IF X$=CHR$(8) AND X>0 THEN X=X-1
140 IF X$=CHR$(10) AND Y<199 THEN Y=Y+1
150 IF X$=CHR$(11) AND Y>0 THEN Y=Y-1
160 IF X$=CHR$(9)AND X<239 THEN X=X+1
170 IF X$="0" OR X$="1" THEN T=VAL(X$):CLS:?:?"TRAIT="T
180 IF X$=CHR$(127) THEN RUN
190 GOTO 40
```

Explications du programme:

— 20: X et Y sont les coordonnées du curseur, T est le paramètre de couleur (1=écrit, 0=efface).

— 40: Cette ligne déplace le curseur, tout en l'effaçant le cas échéant de son ancienne position si T=0.

— 50: Saisie de la touche pressée.

— 60: Mise en mémoire dans XM et YM des valeurs de X et Y, avant leur modification éventuelle.

— 70: Teste si la touche SHIFT est appuyée (pour l'explication de cette variable-système, voir § 2.3.3).

— 80-110: Gestion des déplacements en diagonale; incrémente ou décrémente X et Y en fonction de la touche appuyée en même temps que SHIFT et en veillant à ne pas dépasser les limites de l'écran.

— 130-160: Gère les déplacements simples (haut, bas, droite, gauche).

— 170: Changement du mode curseur (écrit-efface), et l'affiche dans la fenêtre-texte.

180: Touche DEL, effacement écran.

Dessiner un rectangle

```
10 HIRES:C1=100:C2=50:X=60:Y=60
20 CURSET X,Y,3
30 DRAW C1,0,2:DRAW 0,C2,2:DRAW-C1,0,2:DRAW 0,-C2,2
```

— 10: Définition de la taille des côtés (C1 et C2) et de la position du coin supérieur gauche.

Dessiner un rectangle plein

```
10 HIRES:C1=100:C2=50:X=60:Y=60
20 CURSET X,Y,3:FOR N=1 TO C2
30 DRAW C1,0,2:CURMOV-C1,1,3
40 NEXT
```

Faites tourner ce programme, et ajoutez ces deux lignes:

```
15 CURSET 100,100,3:CIRCLE 50,1
```

et

```
50 IF X=40 THEN END ELSE X=40:Y=80:GOTO 20
```

Cela vous donne un aperçu des possibilités du graphique !

Dessiner une ellipse

```
 5 REM  ELLIPSE
 7 REM
10 INPUT"Rayons horizontal, vertical";R1,R2
20 INPUT"Coordonnées du centre (X,Y)";X,Y
30 HIRES
40 S=PI/(R1+R2)/1.8
50 FOR I=0 TO 2*PI STEP S
60 X1=SIN(I)*R1+X
70 Y1=COS(I)*R2+Y
80 CURSET X,Y,1
90 NEXT I
```

— 40: Définit le pas séparant chaque point tracé. Si vous voulez un tracé plus fin, augmentez la valeur 1.8 sinon la diminuer.

— 60,70: Calcul de l'abscisse et de l'ordonnée de chaque point de l'ellipse, I étant exprimé en radians. Les deux rayons fournissent un coefficient différent pour chaque coordonnée, déterminant l'aspect elliptique.

"END"

Ce petit programme amusant dessine le mot "END" en caractères énormes, et en utilisant à fond la commande DRAW:

```
 10 REM  PROGRAMME END
 20 REM
100 HIRES
110 CURSET 16,170,3
115 REM "E"
120 DRAW  0,-120,1:DRAW  60,0,1:DRAW 0,16,1:DRAW-44,0,1:
    DRAW  0,36,1:DRAW 30,0,1
130 DRAW  0,16,1:DRAW-30,0,1:DRAW 0,36,1:DRAW 44,0,1:
    DRAW  0,16,1:DRAW-60,0,1
150 REM "N"
160 CURMOV  75,0,3:DRAW 0,-120,1:DRAW 16,0,1:DRAW 28,80,1:
    DRAW  0,-80,1:DRAW 16,0,1
170 DRAW 0,120,1:DRAW-16,0,1:DRAW-28,-80,1:DRAW 0,80,1:
    DRAW  -16,0,1
190 REM "D"
200 CURMOV 75,0,1
210 DRAW  0,-120,1:DRAW 45,0,1:DRAW 10,5,1:DRAW 5,10,1:
    DRAW 0,90,1:DRAW-5,10,1
220 DRAW-10,5,1:DRAW-45,0,1:CURMOV 16,-16,3:DRAW 0,-88,1:
    DRAW 24,0,1:DRAW 5,2,1
230 DRAW 2,5,1:DRAW 0,74,1:DRAW-2,5,1:DRAW-5,2,1:DRAW-24,0,1
```

Aucune explication n'est nécessaire, nous remarquerons simplement que les arrondis de la lettre "D" sont obtenus par approximation, en traçant des petits segments de droite.

La Hardcopy graphique

Nous avons déjà proposé un programme effectuant la copie de l'écran texte sur imprimante, au paragraphe 3.4.1. Cette fois-ci il s'agit d'obtenir une copie du graphique, opération légèrement plus complexe. La méthode de mémorisation de l'écran haute-résolution ayant été expliquée au paragrahe 4.2, nous ne reviendrons pas dessus. Il nous suffira de savoir que la plupart des imprimantes utilisent le même procédé pour l'impression point par point, à savoir qu'un octet contient toujours un groupe de points.

Cependant, deux difficultés sont à résoudre:

— Sur l'écran, les octets sont disposés "horizontalement", alors que sur l'imprimante ils le sont "verticalement", le bit de poids faible étant en

bas. La solution est assez simple, il suffira d'imprimer le graphique "couché", c'est-à-dire incliné de 90°. L'écran sera lu de haut en bas mais de droite à gauche, en commençant par le coin supérieur droit.

— Les imprimantes utilisent les 8 bits de l'octet pour le codage, l'ORIC seulement 6. A chaque ligne imprimée (en fait, une colonne de l'écran), le saut de ligne sur l'imprimante devra correspondre à six points au lieu de 8.

Le programme a été écrit pour une imprimante EPSON 80 série III, voici un résumé des différents codes qui lui sont propres et que vous pourrez éventuellement adapter pour votre propre machine (ESC est le caractère ESCAPE ou CHR$(27):

ESC;"A";CHR$(N): Définit un espacement ligne de N points.

ESC;"D";CHR$(N);CHR$(0): N est la colonne où commence l'impression. Cette commande sert à centrer le dessin, et n'est pas indispensable.

CHR$(9): Effectue le centrage à la position définie par la commande précédente, et peut également être omis.

ESC;"K";CHR$(N);CHR$(0): ESC"K" passe en mode d'impression point par point. Le nombre N renseigne l'imprimante sur le nombre de points à imprimer sur une ligne (ici 200, puisque l'écran est copié de haut en bas).

CHR$(10): Code de passage à la ligne.

```
  5 REM  HARDCOPY GRAPHIQUE
  7 REM
 10 E$=CHR$(27):CALL #ED01
 20 LPRINT E$;"A";CHR$(6);
 30 LPRINT E$;"D";CHR$(23);CHR$(0)
 40 FOR I=#A000 TO #A027
 50 LPRINT CHR$(9);E$;"K";CHR$(200);CHR$(0);
 60 FOR J=7960 TO 0 STEP -40
 70 DA=PEEK(I+J)-64:IF DA<0 THEN DA=0
 80 PRINT
 90 LPRINT CHR$(DA);
100 NEXT J
110 LPRINT CHR$(10);
120 NEXT I
130 CALL #E804
```

Explications:

— 10: Le caractère Escape est attribué à E$, et on désactive le scanning clavier pour éviter les caractères parasites (cf. § 2.4.1).

— 40 : Démarrage d'une boucle de 40 pas (une par ligne).

— 50 : Initialisation de l'imprimante avant lecture-impression de la ligne.

— 60 : Démarrage d'une boucle de 200 pas, nombre d'octets "verticaux" sur chaque ligne.

— 70 : Lecture de l'adresse définie par I+J et chargée dans DA, qui est mis à 0 s'il s'agit d'un attribut (non imprimable).

— 80 : Le PRINT est indispensable pour le bon fonctionnement du programme.

— 90-120 : Envoi de l'octet lu à l'imprimante, saut de ligne à la fin de la boucle interne et suite de la boucle externe.

— 130 : Le clavier est remis en fonction.

4.5. Dessin assisté par ordinateur (D.A.O.) : Le programme MULTIGRAF

Ce programme, utilisant à fond les diverses commandes graphiques de l'ORIC, vous permettra de vous servir de l'écran comme d'une tablette de dessin. Le tracé de droites et de diverses figures géométriques y est simplifié, ainsi que l'usage des divers attributs disponibles (couleur, etc.).

Son principe d'utilisation est assez simple :

— Les déplacements élémentaires du curseur, haut bas et droite gauche, sont obtenus avec les quatre flèches.

— Les déplacements en diagonale, en pressant la touche SHIFT en même temps que l'une des quatre flèches.

— Deux vitesses de déplacement du curseur sont possibles, la vitesse lente étant point par point et la rapide par dix points.

— Le curseur peut écrire, effacer ou ne pas modifier les points sur lesquels il passe.

— Plusieurs commandes utilisent deux paramètres : Pour une droite par exemple, il faut "pointer" les deux extrémités. En fait, il suffit de placer

le curseur à une des extrémités et de presser la lettre correspondant à la commande ; ensuite, on déplace le curseur à la deuxième extrémité et on rappuie sur la lettre-clé ce qui a pour effet d'exécuter la fonction demandée.

Commandes disponibles

→	: Curseur à droite
←	: Curseur à gauche
↑	: Curseur vers le haut
↓	: Curseur vers le bas
→ + **SHIFT**	: Curseur à droite vers le bas
← + **SHIFT**	: Curseur à gauche vers le haut
↑ + **SHIFT**	: Curseur à droite vers le haut
↓ + **SHIFT**	: Curseur à gauche vers le bas
<	: Sélectionne la vitesse de déplacement lente du curseur.
>	: Sélectionne la vitesse rapide.
0,1,2	: Choix du mode curseur, 0=Efface 1=Écrit et 2=Inverse.
<**A**>ide	: Affiche la liste des commandes acceptées par le programme.
<**D**>roite	: Trace une droite entre les deux points marqués par le curseur.
<**R**>ectangle	: Dessine un rectangle dont les angles opposés sont les points marqués par le curseur.
<**C**>ercle	: Dessine un cercle, dont le premier point marqué est le centre et le deuxième un point quelconque de la périphérie.
<**B**>oule	: Trace un cercle plein, avec les mêmes paramètres que pour un cercle normal.
<**Z**>one	: Le premier appui sur "Z" marque un angle ; le deuxième marque l'angle opposé, et le programme vous demande quel attribut vous désirez affecter à cette zone (couleur, motif etc... voir § 4.3.1).

<**T**>rait : Choix du type de trait utilisé pour les dessins. (Voir instruction PATTERN, § 4.3.1).

<**E**>crit : Autorise l'écriture de texte à l'écran. Après chaque caractère tapé le curseur avance automatiquement de six points, se mettant ainsi à la position correcte pour le caractère suivant. Mode annulé par la touche "DEL".

<**P**>rint : Effectue .une copie de l'écran sur imprimante EPSON 80 série III (Hardcopy).

ESCAPE : Donne accès à quatre utilitaires:

<**E**>ffacer l'écran.
<**S**>ave Sauvegarde de l'écran sur cassette.
<**L**>oad Chargement d'un écran se trouvant sur cassette.
<**Q**>uitter le programme.

De plus, les trois lignes de texte en bas d'écran affichent la commande en cours d'exécution, le mode curseur, le type de trait, la vitesse sélectionnée et les coordonnées du curseur.

Ce que nous avons dit sur les nombreuses REM pour les programmes précédents reste valable, à savoir que vous pouvez les éliminer mais surtout gardez les numéros de ligne multiples de 100 à cause des GOTO et GOSUB qui y font référence ainsi que la ligne N° 1.

```
1 REM   MULTIGRAF  DAO
2 REM
4 REM
6 REM   GESTION POINTEURS
8 REM
10 IFA(1)=0THEN DOKE#400,DEEK(#9C):GOTO100
20 IFDEEK(#9C)<>DEEK(#400)THENDOKE#9C,DEEK(#400):GOSUB6500
30 GOSUB6200:GOTO1000
100 CLS:HIMEM38000:POKE775,32
108 REM
110 REM    PROGRAMME
112 REM
120 GOSUB8000:E$=CHR$(27)
130 HIRES
140 GOSUB7000:GOSUB6200
150 CURSETX,Y,1
998 REM
```

```
1000 REM   BOUCLE PRINCIPALE
1002 REM
1100 GET X$:AX=ASC(X$)
1170 IFAX>7ANDAX<12THENGOSUB2000:GOTO1100  'Mouvements curseur
1175 REM
1180 IFAX>47ANDAX<51THEN T=VAL(X$):GOTO1500'Chgt mode curseur
1185 REM
1190 IFX$="T"THENGOSUB4500'Pattern
1200 IFAX=27THEN1600       'Utilitaires
1210 IFX$="R"THENGOSUB3000'Rectangle
1220 IFX$="D"THENGOSUB3200'Droite
1230 IFX$="Z"THENGOSUB3400'Zone-Fill
1240 IFAX=127ANDZ=-1THENCURSETX,Y,2:X=XM:Y=YM:GOSUB6400 'Annulation
1250 IFX$="E"THENGOSUB4000'Texte
1260 IFX$="C"THENGOSUB3600'Cercle
1270 IFX$="B"THENGOSUB3800'Boule
1280 IFX$="A"THENGOSUB6000'Aide
1281 REM
1282 REM
1285 REM   Vitesse deplacements
1287 REM
1290 IFX$=","THENV$="<Lente ":V=1:POKE775,35
1300 IFX$="."THENV$=">Rapide":V=10:POKE775,32
1305 REM
1310 IFX$="P"THENGOSUB5000'Hardcopy
1500 GOSUB6200:GOTO1100
1598 REM
1600 REM   UTILITAIRES
1602 REM
1610 CLS
1620 PRINT"<E>ffacer l'ecran, <Q>uitter le prog."
1625 PRINT"<S>ave ou <L>oad un ecran,"
1630 PRINT"ou bien n'importe quelle touche.";
1640 GETX$
1650 IFX$="E"THENHIRES:PRINTCHR$(17);:GOTO1500
1148 IFX$="Q"THEN TEXT:END
1670 IFX$="S"THEN1700
1680 IFX$="L"THEN1800
1690 GOTO1500
1698 REM
1700 REM   Save ecran
1702 REM
1710 CLS:POKE618,7
1720 INPUT"NOM DE L'ECRAN ";NN$
1730 PRINT"Lancez le K7 et pressez une touche"
1740 GETX$
1750 CSAVENN$,A#A000,E#BF3F
1760 GOSUB6500:GOTO1500
1798 REM
1800 REM   Load ecran
1802 REM
1810 CLS:POKE618,7
```

```
1820 INPUT"NOM DE L'ECRAN ";NN$
1830 PRINT"IMPORTANT: Des que l'ecran est charge,";
1840 PRINT"tapez GOTO 1 pour relancer le prog.";
1850 CLOADNN$,A#A000,E#BF3F
1860 GOTO1500
1998 REM
2000 REM   Mouvements curseur
2002 REM
2003 REM   Haut/Bas/Droite/Gauche
2004 REM
2010 CURSETX,Y,T
2015 IF PEEK(521)<>56THEN2100
2020 IFAX<>8THEN2030
2022 IFX-V>=0THENX=(X-V)ELSEX=MX
2024 GOTO2060
2030 IFAX<>9THEN2040
2032 IFX+V<MXTHENX=(X+V)ELSEX=0
2034 GOTO2060
2040 IFAX<>10THEN2050
2042 IFY+V<MYTHENY=(Y+V)ELSEY=0
2044 GOTO2060
2050 IFY-V>=0THENY=(Y-V)ELSEY=MY
2060 CURSETX,Y,2
2070 GOSUB6200:RETURN
2098 REM
2100 REM   Diagonales
2102 REM
2110 IFAX<>8THEN2120
2112 IFX-V>=0THENX=(X-V)ELSEX=MX
2114 IFY-V>=0THENY=(Y-V)ELSEY=MY
2116 GOTO2060
2120 IFAX<>9THEN2130
2122 IFX+V<MXTHENX=(X+V)ELSEX=0
2124 IFY+V<MYTHENY=(Y+V)ELSEY=0
2126 GOTO2060
2130 IFAX<>10THEN2140
2132 IFX-V>=0THENX=(X-V)ELSEX=MX
2134 IFY+V<MYTHENY=(Y+V)ELSEY=0
2136 GOTO2060
2140 IFX+V<MXTHENX=(X+V)ELSEX=0
2142 IFY-V>=0THENY=(Y-V)ELSEY=MY
2144 GOTO2060
2999 REM
3000 REM   Rectangle
3002 REM
3010 GOSUB 6400
3020 IFZ=-1THENXM=X:YM=Y:M$="LRectgle":RETURN
3030 IFT=2THENCURSETXM,YM,2:CURSETX,Y,2
3040 DRAWXM-X,0,TM:DRAW0,YM-Y,TM:DRAWX-XM,0,TM:DRAW0,Y-YM,TM
3050 IFT=1THENRETURN
3060 X=X+SGN(XM-X):Y=Y+SGN(YM-Y):CURSETX,Y,1
3070 RETURN
3199 REM
```

```
3200 REM   Droite
3202 REM
3210 GOSUB 6400
3220 IFZ=-1THENXM=X:YM=Y:M$="LDroite":RETURN
3240 DRAWXM-X,YM-Y,TM
3250 RETURN
3399 REM
3400 REM   Zones
3402 REM
3410 GOSUB6400
3420 IFZ=-1THENXM=X:YM=Y:M$="LZone":RETURN
3430 CURSETX,Y,2:CURSETXM,YM,2
3440 RX=INT(ABS((XM-X)/6)):IFXM<XTHENX=XM
3445 IFRX<1THENRX=1
3450 RY=ABS(YM-Y):IFYM<YTHENY=YM
3455 IFRY=0THENRY=1
3460 CURSETX,Y,2
3500 CLS:POKE618,7
3510 PRINT"0-7:Coul.Aff 12:Flash 16-23:Coul.Fond"
3520 PRINT"32-127:Dessin de la Zone"
3530 INPUT"VOTRE CHOIX ";FI
3540 IF FI>7ANDFI<12ORFI>12ANDFI<16ORFI>23ANDFI<32ORFI>127THEN3500
3550 FILLRY,RX,FI
3560 RETURN
3599 REM
3600 REM   Cercle
3602 REM
3610 GOSUB6400
3620 IFZ=-1THENXM=X:YM=Y:M$="LCercle":RETURN
3640 RD=SQR((XM-X)^2+(YM-Y)^2)
3650 X=XM:Y=YM:CURSETX,Y,1
3660 CIRCLERD,TM
3670 RETURN
3698 REM
3800 REM   Boule
3802 REM
3810 GOSUB6400
3820 IFZ=-1THENXM=X:YM=Y:M$="LBoule":RETURN
3840 RD=SQR((XM-X)^2+(YM-Y)^2)
3850 X=XM:Y=YM:CURSETX,Y,1
3860 FORN=2TORD
3870 CIRCLEN,TM
3880 NEXTN
3890 RETURN
3998 REM
4000 REM   Ecrire a l'ecran
4002 REM
4010 CLS:PRINT:M$="LTexte"
4020 T=2:V=6:MX=230:MY=190:V$="Rapide"
4030 GOSUB6200
4040 X$=KEY$:IFX$=""THEN4040ELSEAX=ASC(X$)
4050 IFAX=127THENRESTORE:GOSUB7000:POKE#20C,255:RETURN
4060 IFAX=20THENPRINTCHR$(20);:GOTO4040
```

```
4070 IFAX<32THENGOSUB2000:GOTO4040
4090 CHARAX,0,T
4100 AX=9:GOSUB2000
4110 GOSUB6200
4200 GOTO4040
4499 REM
4500 REM   Choix du trait
4502 REM
4510 CLS:POKE618,7
4520 PRINT"CHOIX DU TYPE DE TRAIT-"
4530 PRINT"VALEURS AUTORISEES: 0-255
4540 INPUT"Votre choix:";P
4550 IFP<0ORP>255THEN4510
4560 PATTERNP:RETURN
4998 REM
4999 REM
5000 REM   HARDCOPY
5002 REM
5010 CLS
5020 PRINT"ALLUMEZ L'IMPRIMANTE"
5030 PRINT"<R>etour au pgm, ou pressez une touche";
5040 GETX$:IF X$="R"THEN RETURN ELSE CALL#ED01
5100 LPRINT E$"A"CHR$(6);
5110 LPRINT E$"D"CHR$(23)CHR$(0);
5120 FOR I=#A000 TO #A027
5130 LPRINT CHR$(9)E$"K"CHR$(200)CHR$(0);
5140 FOR J=7960 TO 0 STEP-40
5150 DA=PEEK(I+J)-64:IF DA<0THENDA=0
5160 PRINT
5170 LPRINT CHR$(DA);
5180 NEXTJ
5190 LPRINT CHR$(10);
5200 NEXTI
5210 CALL#E804:RETURN
5997 REM
5998 REM   MESSAGES EN BAS D'ECRAN
5999 REM
6000 REM   Aide
6005 CLS
6010 PRINT"<R>ectangle <D>roite <C>ercle <B>oule"
6020 PRINT"<Z>one <T>rait <A>ide <E>crit <P>rint"
6030 PRINT"<DEL>:Anul <ESC>:Utilit 0,1,2:Curseur";
6050 GETX$
6060 RETURN
6199 REM
6200 REM   Statut
6202 REM
6205 IF PEEK(526)<10 THEN RETURN
6210 CLS:POKE618,6
6220 PRINT"MODE:"E$M$E$"HCURS:"T$(T)" TRAIT:"P
6230 PRINT"VITESSE:"V$"  "E$"AX="X"Y="Y;
6240 RETURN
```

```
6399 REM
6400 REM   Memorisation coordonnees
6402 REM
6410 Z=NOTZ
6420 IFZ=-1THENXM=X:YM=Y:TM=T:T=2:CURSETXM,YM,2:RETURN
6430 T=TM:M$="@Deplct"
6440 RETURN
6498 REM
6500 REM   Vidage ligne 0 (SAVE-LOAD)
6502 REM
6510 FORN=0TO28
6520 POKE#BB80+N,64
6530 NEXT
6540 RETURN
6999 REM
7000 REM   INITIALISATIONS
7002 REM
7010 READ X,Y,T,P,V,MX,MY,A(1)
7020 READ M$
7030 READT$(0),T$(1),T$(2)
7040 READV$
7200 RETURN
7498 REM
7500 REM   Donnees de depart
7502 REM
7510 DATA 100,100,2,255,10,239,199,1
7520 DATA @Deplct
7530 DATA Efface,Ecrit,Inverse
7540 DATA >Rapide
7997 REM
7998 REM--------FIN DU PROG----------
7999 REM
8000 REM   **INTRODUCTION**
8002 REM
8010 E$=" "+CHR$(27)
8100 PRINTCHR$(17)CHR$(4):DOKE#BBD0,2578:DOKE#BBF8,2578
8110 PRINT"MULTIGRAF Programme de D.A.O  "CHR$(96)" BVRP"
8120 PRINTCHR$(4):PRINT
8130 PRINT" Ce programme vous permettra d'exploi-";
8140 PRINT"ter les possibilites graphiques d'ORIC";
8150 PRINT"avec le maximum de confort.":PRINT
8160 PRINT"DEPLACEMENTS CURSEUR:"
8170 PRINT"4 fleches: Horizontal/Vertical"
8180 PRINT"4 fleches+SHIFT:Diagonales":PRINT
8190 PRINT"VITESSE DEPLCT:":PRINT"< Lente, point par point"
8200 PRINT"> Rapide, 10 points par 10 points":PRINT
8210 PRINT"Les autres commandes s'obtiennent par"
8220 PRINT"leur initiale:<D>roite,<C>ercle,etc."
8230 PRINT"<A>ide donne la liste des commandes"
8240 PRINT"acceptees.":PRINT
8250 PRINT"L'ecran affiche la commande en cours"
8260 PRINT"d'execution, le mode curseur (ecrit,"
```

```
8270 PRINT"efface,inverse) et sa position."
8280 PRINT"Annulation d'une commande: DEL":PRINT
8300 PRINTE$"L  <D>etails ou <P>rogramme";
8310 GETX$:IFX$="P"THENPRINTCHR$(17):CLS:RETURN
8320 IFX$<>"D"THEN8310
8400 CLS:POKE48080,22
8410 PRINT:PRINT"       EXPLICATIONS DETAILLEES":PRINT
8420 PRINT" Commandes necessitant 2 parametres:":PRINT
8430 PRINT"-Un premier appui sur la lettre-cle"
8440 PRINT"marque la 1ere position,materialisee"
8450 PRINT"par un point.Ensuite, on deplace le"
8460 PRINT"curseur a la 2eme position,et on tape"
8470 PRINT"a nouveau la lettre-cle.":PRINT
8480 PRINT"EX: Les 2 extremites d'une droite,les"
8490 PRINT"diagonales d'un rectangle,le centre et";
8500 PRINT"un point du perimetre pour les cercles";
8510 PRINT"et les boules.":PRINT
8520 PRINT" Commandes n'exigeant qu'un parametre:"
8530 PRINT"-Celles-ci sont executees immediate-"
8540 PRINT"ment, l'ecran vous renseigne sur les"
8550 PRINT"valeurs autorisees.":PRINT
8560 PRINT"La commande <P>rint recopie l'ecran"
8570 PRINT"sur une imprimante EPSON serie 80.":PRINT:PRINT
8580 PRINTE$"LPOUR COMMENCER PRESSER UNE TOUCHE";
8590 GETX$:PRINTCHR$(17);:RETURN
9000 CLS
9010 PRINTPEEK(526);
9020 WAIT 10:GOTO9010
```

Explications du programme

— 10-30 : Mise en mémoire des pointeurs de fin de programme, pour restitution dans le cas du chargement d'un écran. Comme dans ce cas on est obligé de relancer manuellement le programme par GOTO 1, la ligne 30 permettra de s'y rendre directement en sautant l'introduction.

— 100 : POKE 775,32 choisit une vitesse d'autorepeat plus rapide que la normale, afin d'accélérer les mouvements curseur.

— 120 : Si vous voulez éviter de taper les lignes 8000-fin, qui ne contiennent que des explications, il faudra aussi supprimer le GOSUB 8000 de cette ligne.

— 1000-1280 : Boucle saisissant la touche frappée (X$), et effectuant le branchement vers les divers sous-programmes concernés. AX est le code ASCII de X$ et T sa valeur s'il s'agit d'un nombre ; XM et YM mémorisent les coordonnées du curseur, avant son déplacement. Z est un pointeur valant 1 en temps normal et —1 après pointage des premières coordonnées d'une figure, stockées dans XM et YM.

— 1290: Modification de la vitesse de déplacement (variable V, valant 1 ou 10).

— 1600-1860: Utilitaires; Effacement de l'écran, sauvegarde et relecture cassette. NN$ est le nom de l'écran à enregistrer ou relire.

— 2000-2070: Gestion des mouvements simples du curseur, avec vérification de sa position: s'il dépasse les limites de l'écran, il repasse de l'autre côté de celui-ci. La variable T détermine si le curseur doit être effacé ou non de son ancienne position.

— 2100-2144: Gestion des mouvements curseur complexes (diagonales). Identique à ci-dessus, avec la différence que chaque fois X et Y sont testées et incrémentées ou décrémentées ensemble.

— 3000-3070: Sous-programme dessinant un rectangle, dont les angles opposés ont les coordonnées XM,YM et X,Y. Le drapeau Z détermine si on pointe le premier ou le deuxième lot de coordonnées. TM stocke l'ancienne valeur de T.

— 3200-3250: Trace une droite entre XM,YM et X,Y.

— 3400-3560: Définition d'une zone. Comme la coordonnée horizontale de celle-ci est en nombre de cellules et non pas de points, la ligne 3440 opère la conversion et RX contient la valeur trouvée. Au deuxième passage dans le sous-programme, la valeur de l'attribut ou motif à y placer est demandée et un petit mémo des principale valeurs affiché en bas d'écran.

— 3600-3670: Trace un cercle; son rayon (RD) est calculé à la ligne 3640 à l'aide du célèbre théorème de pythagore (Le carré de l'hypoténuse, ici le rayon, est égal à la somme des carrés des deux côtés opposés XM-X et YM-Y).

— 3800-3890: Dessine une boule, en traçant tous les cercles dont le rayon est compris entre 1 et le rayon demandé.

Note: Si un cercle ou une boule déborde de l'écran, une erreur sera provoquée et le programme s'arrêtera. Il suffira alors de le relancer par GOTO 1.

— 4000-4200: Gestion de l'écriture de texte. Le mode curseur inverse est imposé, car la seule manière d'effacer un caractère est de réécrire par-dessus dans ce mode. Il en va de même pour le nombre de points sautés en déplacement, qui est de six afin de respecter l'espacement entre caractères. A la ligne 4050, l'instruction POKE#20C,255 force

les majuscules si on quitte le mode afin de permettre la saisie des caractères de commande.

— 4500-4520 : Sélection du trait utilisé pour les tracés (variable P, argument d'une instruction PATTERN).

— 5000-5210 : Copie l'écran sur une imprimante EPSON. Ce programme est présenté avec des explications détaillées au paragraphe 4.4.

— 6000-6060 : Affichage du petit texte d'aide, résumant les commandes acceptées.

— 6200-6240 : Affichage des paramètres en vigueur, dans les trois lignes en bas d'écran. M$ est la commande en cours d'exécution, ou alors "déplacement". Le tableau T$( ) est "Écrit", "Efface" ou "Inverse" en fonction de la valeur de T. V$ est la vitesse de déplacement, puis viennent X et Y. Ce sous-programme est très fréquemment appelé, en fait à chaque passage dans la boucle principale. L'accès en est interdit uniquement en autorepeat, pour ne pas ralentir le curseur.

— 6400-6440 : Autre routine appelée par chaque commande, qui charge XM et YM au pointage de la première paire de coordonnées et gère le drapeau Z.

— 6500-6540 : Lors du chargement ou de la sauvegarde d'un écran, les messages "Saving...XXX" ou "Loading...XXX" s'affichent à la ligne 0, mais ils apparaissent en tant que motif graphique au lieu de caractères. Ensuite, le message disparaît mais les octets qu'il occupait demeurent inaccessibles pour dessiner, situation que cette routine a pour tâche de corriger.

— 7000-7540 : Attribue leur valeur initiale à toutes les variables du programme. Au départ, le curseur est positionné à X=100,Y=100 en mode inverse et vitesse rapide. MX et MY sont les valeurs maximales permises pour X et Y, respectivement 239 et 199.

— 8000-Fin : Introduction fournissant toutes les explications nécessaires à l'utilisation immédiate de MULTIGRAF mais que vous pouvez très bien omettre si les travaux de frappe vous rebutent. Dans ce cas, ne manquez pas de corriger également la ligne 120 qui appelle ce sous-programme.

Précisons pour terminer que le programme occupe 6394 octets en mémoire tel qu'il est présenté, valeur qui passe à 5583 sans les REM et à 3900 octets (!) si le texte d'introduction est également supprimé.

5

Le son ORIC

Muni d'une "puce" spécialisée chargée de la gestion du son, l'ORIC présente de ce fait des dispositions nettement au-dessus de la moyenne dans ce domaine. Outre un certain nombre de sons pré-programmés tout indiqués pour les jeux, des notes ou du bruit blanc peuvent être générés sur trois canaux différents à l'aide d'instructions particulières.

Ces trois canaux peuvent produire du son de fréquence déterminée, ou alors des notes de la gamme dont la hauteur et l'enveloppe sont programmables. Le bruit blanc, s'il est utilisé, sera mélangé à une des trois sorties.

5.1. Principes de fonctionnement

Pour bien comprendre comment exploiter le son sur l'ORIC, il faut savoir qu'en fait les instructions spécialisées MUSIC et SOUND produisent un

son conditionné, dans une certaine mesure, par la nature du son les ayant précédé: Ainsi, il sera impossible d'obtenir une note pure après un EXPLODE (qui ne contient quasiment que du bruit blanc). D'une manière générale:

— Si le bruit blanc est utilisé sur un canal il faudra ensuite l'"éliminer" par un son prédéfini qui n'en contient pas, comme par exemple la répétition sonore du clavier ou CTRL G.

— Il faudra tenir compte du fait qu'on n'aura pas nécessairement le même résultat en exécutant deux fois de suite la même instruction à des endroits différents d'un programme, certaines modifications apportées au son de départ restant actives après changement des paramètres qui les ont créées.

— Les sons continus durent indéfiniment sauf avis contraire. Ils seront interrompus, par l'émission d'un autre son quel qu'il soit, ou par l'instruction PLAY 0,0,0,0.

— Enfin, vous pourrez jouir à loisir de vos créations sonores en connectant l'ORIC à votre chaîne Hi-Fi, par l'intermédiaire de la prise DIN 7 broches du magnétocassette. Le son est disponible entre la broche 5 ou la broche 4 et la masse (broche 2), avec un niveau de 40 millivolts environ au volume maximum.

5.2. *Commandes disponibles*

Les sons pré-programmés

PING : Son cristallin, identique à la sonnerie clavier (CTRL G ou CHR$(7)).

ZAP : Sifflement du type "canon laser".

SHOOT : Bruit simulant un coup de feu — contient du bruit blanc.

EXPLODE: Bruit d'explosion — contient du bruit blanc.

Et bien sûr, le "clic" produit par le clavier, qui ne peut pas être obtenu par programme (mais qui se simule avec l'instruction PLAY, voir plus bas).

Instructions produisant un son

SOUND

Syntaxe: SOUND <canal>,<N>,<volume>

Application: Génère un son continu avec les paramètres suivants:

Canal: 1, 2 ou 3 produira un son pur seul sur le canal spécifié;
4, 5 ou 6 mélangera un bruit avec le son d'un des trois canaux (4 avec le canal 1, etc...).

N: Détermine la hauteur du son, qui peut être trouvée par la formule suivante:

N=62000/ < fréquence >

Cette équation est valable pour des valeurs de N allant de 1 (62 kHz) à 4000 environ (15 Hz).

Volume: Modifiable par pas fixes, de 1 à 15. La valeur 0 donne la main à l'instruction PLAY, définissant l'enveloppe du son (voir ci-après).

Exemple: SOUND 1,62,9 produira un son continu à 1000 Hz.

MUSIC

Syntaxe: MUSIC <Canal>,<Octave>,<Note>,<Volume>

Application: Génère une note répondant aux paramètres fournis qui sont:

Canal: Ne peut prendre que les valeurs 1,2 ou 3.

Octave: Entier compris entre 0 (octave la plus basse) et 6 (octave la plus haute).

Note: Entier compris entre 1 et 12, correspondant aux notes suivantes:

1	— DO	7	— FA#
2	— DO#	8	— SOL
3	— RE	9	— SOL#
4	— RE#	10	— LA
5	— MI	11	— LA#
6	— FA	12	— SI

Volume: Choix du niveau, comme pour SOUND.

PLAY

Syntaxe: PLAY <Son>,<Bruit>,<Enveloppe>,<Durée>

Application: PLAY *ne produit pas* de son, mais modifie un son prédéfini ou généré par SOUND et MUSIC, lorsque le volume est mis à 0 dans ces instructions.

Son: Sélection des canaux qui émettront un son pur, de 0 à 7:

Valeur	*Canaux activés*
0	Aucun
1	1
2	2
3	1 + 2
4	3
5	1 + 3
6	2 + 3
7	1 + 2 + 3

Bruit: Sélectionne les canaux produisant du bruit blanc, avec les mêmes codes que pour <Son>.

Enveloppe: De 1 à 7, choix du "contour" d'une note ou d'un son.

Numéro	*Caractéristiques*
1	Son unique, avec attaque nette et decay lent.
2	Son unique, attaque lente et arrêt net.
3	Comme 1, mais répétitif.
4	Répétitif, avec attaques et decays lents.
5	Une attaque nette, puis son continu.
6	Comme 2, répétitif.
7	Une attaque lente, puis son continu.

Durée: Choix de la longueur du son conformé par l'enveloppe, peut avoir une valeur entière comprise entre 1 et 37000. Agit de trois manières différentes:

— S'il s'agit d'une enveloppe spécifiant un son unique (N° 1 et 2), détermine la durée de celui-ci.

— Pour une enveloppe produisant un son répétitif (N° 3, 4 et 6), définit l'intervalle entre deux répétitions. Donc si cet intervalle est suffisamment court, il devient lui-même une fréquence.

— S'il s'agit d'un son continu (enveloppes 5 et 7), définit simplement la longueur de l'attaque.

  N étant la longueur souhaitée en millisecondes, on trouvera <durée> avec la formule suivante:

  < Durée > =N(ms)*2

Exemples:

Pour voir comment on peut produire un son avec PLAY sans utiliser au préalable MUSIC ou SOUND, taper:

```
PLAY 0,7,3,120
```

Puis:

```
SHOOT:PLAY 0,7,3,120
```

On obtient deux bruits distincts (rappelant un hélicoptère) avec la même instruction, simplement en "travaillant" des sons de départ différents: La répétition sonore du clavier pour la première ligne, et SHOOT pour la deuxième.

Simulation du "clic" de clavier par programme:

```
PLAY 7,0,1,20
```

On remarque que cette instruction reproduit fidèlement le dernier "clic" entendu, "RETURN" et les flèches ne produisant pas exactement le même que le restant du clavier.

5.3. Une application: le programme MONOSYNTH

Il est extrêmement difficile de véhiculer l'impression de sons ou d'instructions qui les gèrent par des mots, nous avons donc conçu le petit programme ci-dessous qui vous donnera la possibilité de vous servir du clavier

comme d'un instrument mais surtout celle d'agir sur tous les paramètres définissant un son.

Ces derniers sont affichés à l'écran, avec les noms et les valeurs qu'ils ont dans les instructions MUSIC et PLAY. Ainsi, vous pourrez exactement savoir "qu'est-ce qui fait quoi", et éventuellement retenir les combinaisons les plus intéressantes pour d'autres applications.

Présentation de l'écran, de haut en bas

— Une ligne contient les noms des cinq paramètres principaux: Canal et Octave, arguments de MUSIC, puis Bruit, Durée et Enveloppe, arguments de PLAY.

— Juste en dessous sont affichées les valeurs de chacun des paramètres. L'une d'entre elles clignote, cela veut dire qu'on peut la modifier: Elle augmente d'une unité à chaque appui sur "flèche vers le haut", et diminue avec la flèche vers le bas. On passe d'un paramètre à l'autre avec les flèches droite et gauche.

— Deux lignes plus bas, on trouve la liste des touches du clavier produisant une note: Il s'agit de la rangée supérieure, de "1" à "=", et encore en dessous leur valeur dans la gamme. Chaque fois qu'une note est jouée, un petit caractère spécial en forme de note s'affiche en dessous.

— Un petit texte d'aide en bas d'écran termine la présentation.

Une dernière remarque: Si vous voulez adapter le programme pour pouvoir mémoriser des mélodies et les rejouer ensuite, vous voudrez également les sauvegarder sur cassette, en utilisant un tableau pour stocker les données Dans ce cas, n'oubliez *surtout pas* de charger en mémoire la routine "sauvegarde et relecture de variable" donné au paragraphe 2.4.2 avant de commencer à taper le programme.

```
  1 REM  ** MONOSYNTH **
  2 REM
 10 CLS:GOSUB 1000
 18 REM
 20 REM  PRESENTATION ECRAN
 22 REM
100 ?:?" "CHR$(4)CHR$(27)"J      MONOSYNTH @ @ @"CHR$(4)
110 ?:?:?:PLOT 0,5,17:PLOT 0,9,17
120 ?"CANAL OCTAVE BRUIT DUREE ENVELOPE"
130 POKE 775,25:POKE 616,18:?
134 REM
135 REM  AIDE
136 REM
140 ? E$"S<--> Selection fonction "E$"W"
150 ? E$"S ↑&  Changement valeur "E$"W"
160 ?:? E$"S <Space>×2 Remise à blanc"E$"W"
164 REM
165 REM  IMPRESSION PARAMETRES
166 REM
170 POKE 618,10:F=1:NM=1
180 FOR N=1 TO 5
190 PLOT M(N,3),7,STR$(M(N,0))+"  ":PLOT M(N,3)+6,7,8
200 NEXT:PLOT M(4,3)-1,7,8
220 PLOT M(F,3)-1,7,12:POKE 616,9:?
230 ?" 1     2     3   4     5   6   7     8     9     0   -     =":?
240 ?" DO         RE        MI  FA        SOL       LA        SI"
250 ?"      DO#      RE#           FA#      SOL#     LA#
296 REM
297 REM  BOUCLE PRINCIPALE
198 REM  SAISIE NOTES
299 REM
300 GET X$:AS=ASC(X$):NO=VAL(X$)
310 IF X$="0" THEN NO=10
320 IF X$="-" THEN NO=11
330 IF X$="=" THEN NO=12
340 IF NO=0 THEN 440
350 PLOT NM*3-1,10,32:NM=NO:PLOT NO*3-1,10,64
354 REM
355 REM  JEU NOTES
356 REM
360 ON M(1,0) GOTO 370,380,380
370 MUSIC M(1,0),M(2,0),NO,0:PLAY 1,M(3,0),M(5,0),M(4,0):GOTO 300
380 MUSIC M(1,0),M(2,0),NO,0:PLAY 7,M(3,0),M(5,0),M(4,0):GOTO 300
398 REM
400 REM  SAISIE NOUVEAUX PARAMETRES
402 REM
440 IF AS=32 THEN ?CHR$(6);:GOTO 300
450 IF AS=9 AND F<5 THEN F1=1:GOSUB 900
460 IF AS=9 AND F>1 THEN F1=-1:GOSUB 900
470 IF AS=11 AND M(F,0)<M(F,2) THEN M(F,0)=M(F,0)+1:IF F=4 THEN
    M(F,0)=M(F,0)+9
```

```
480 IF AS=10 AND M(F,0) >M(F,1) THEN M(F,0)=M(F,0)-1:IF F= 4 THEN
    M(F,0)=M(F,0)-9
484 REM
485 REM  IMPRESSION NOUVEAUX PARAM.
486 REM
490 PLOT M(F,3),7,STR$(M(F,0))+"  "
500 GOTO 300
900 PLOT M(F,3)-1,7,8:F=F+1:PLOT M(F,3)-1,7,12:RETURN
999 REM
1000 REM  INITIALISATIONS
1002 REM
1010 FOR N=1 TO 5
1020 READ M(N,0),M(N,1),M(N,2),M(N,3)
1030 NEXT
1040 DOKE 46592,1031:DOKE 46594,1031:DOKE 46596,7172:
     DOKE 46598,6204
1050 DOKE 46384,2056:DOKE 46386,2056:DOKE 46388,5162:
     DOKE 46390,8
1070 E$="  "+CHR$(27):RETURN
1200 DATA 1,1,3,2,3,0,6,9,0,0,6,16,500,0,32767,23,1,1,7,32
```

Explications du programme:

La grande majorité des variables utilisées par le programme sont regroupées dans le tableau à deux dimensions M(X,Y), organisé comme suit:

Y=Paramètre → / *X=Fonction*	*Valeur en cours*	*Val.Min. autorisée*	*Val.Max. autorisée*	*Position horiz. à l'écran*
CANAL	M(1,0)	M(1,1)	M(1,2)	M(1,3)
OCTAVE	2,0	2,1	2,2	2,3
BRUIT	3,0	3,1	3,2	3,3
DURÉE	4,0	4,1	4,2	4,3
ENVELOPE	5,0	5,1	5,2	5,3

— Ce tableau regroupe donc toutes les valeurs affichées à l'écran, et une variable (F) déterminant quelle fonction est en cours de modification permet de tout traiter dans une seule boucle.

— Deux caractères spéciaux sont utilisés, " @ " est transformé en note de musique et " & " en flèche vers le bas.

— 100-160: Affichage du logo en double hauteur, et de tous les textes sur l'écran.

— 170-250 : Impression des valeurs initiales des divers paramètres. PLOT place un code "caractères normaux" (8) après chaque valeur, de telle sorte que le code "clignotant" (12) servant à repérer la fonction active n'agit que sur une d'entre elles à la fois.

— 300-350 : Boucle principale. NO est la valeur de la note jouée, et les lignes 310-330 y placent une valeur correcte pour les touches non numériques. La ligne 340 détecte l'utilisation des flèches, qui sont traitées plus loin. NM mémorise la précédente note jouée, pour effacer le caractère "note" avant de le replacer.

— 360-380 : La note saisie est jouée par une instruction MUSIC et conformée par PLAY, leurs différents paramètres utilisant les valeurs sélectionnées à l'écran.

— 400-480 : Gestion des déplacements latéraux (changement de fonction) et haut/bas (changement de la valeur d'une fonction). F1 est la valeur à ajouter à la valeur en cours du paramètre modifié (1 si on l'augmente et —1 si on le diminue).

— 490-900 : Impression de la nouvelle valeur du paramètre modifié à la position contenue dans M(F,3).

— 1000-1030 : Initialisations : Les valeurs de départ, position d'impression, ainsi que les maxima et minima de chaque paramètre sont chargés à partir de DATA dans le tableau M( ).

— 1040-1050 : Création des deux caractères spéciaux utilisés par le programme.

5.4. N-Bomber

Jeu intéractif du type "jeu d'arcades", N-Bomber n'a pas de relation directe avec le son en particulier. Nous avons toutefois pensé qu'il serait agréable de terminer cet ouvrage par un programme à vocation résolument ludique.

Comme pour tous les logiciels de cette catégorie, la définition en est simple mais la réalisation relativement complexe. Le lecteur patient pourra extraire nombre de "trucs" intéressants en épluchant ce programme qui est, précisons-le, écrit entièrement en Basic.

Le principe du jeu est le suivant : Vous pilotez un bombardier de nuit dans le ciel ennemi, et à vos pieds se déroule un décor composé de caractères semi-graphiques aléatoires ; bombarder ces derniers ne rapporte rien, mais de temps à autre une fusée ou un réservoir apparaissent, qui vous rapportent des points si vous les touchez.

Pour corser la difficulté, la D.C.A. vous canarde sans relâche (et bien souvent vous touche !) et vous consommez trente litres de fuel à chaque tour d'écran sur les 300 que vous avez au départ. Seule, la destruction des réservoirs vous évitera la panne sèche au bout d'un certain temps.

On a le choix entre deux niveaux de difficulté, et tous les 1000 points vous avez droit à un avion supplémentaire (3 au départ), ainsi qu'à huit bombes.

Comme d'habitude, le programme est constitué de sous-programmes chargés de tâches précises, et qui ne se suivent pas forcément dans l'ordre où ils sont appelés. D'une manière générale, les routines servant le plus souvent sont au début, et celles utilisées une seule fois à la fin du programme.

Le score, la quantité de munitions, de fuel et d'avions restants sont affichés en haut de l'écran, ainsi que le High score qui ne s'effacera d'ailleurs pas tant que l'ordinateur n'est pas éteint (ni RUN, ni NEW n'y feront rien).

L'avion change d'altitude avec les flèches haute et basse, et une bombe est larguée en pressant la barre d'espacement. La D.C.A. "suit" l'avion dans ses déplacements verticaux, et tire aléatoirement dans une fourchette de trois cases autour de sa position. Vous pouvez monter au-dessus du plafond de la D.C.A., mais il devient très malaisé de toucher les cibles...

Enfin et pour mémoire, N-Bomber n'occupera que 3,5 Koctets de RAM tel quel et moins de 2,5 K si le thème musical du début et les REM sont supprimés.

```
100 REM   NIGHT  BOMBER
102 REM
104 REM
110 PAPER4:INK0:E1$=CHR$(27):E$=" "+E1$
120 POKE618,14
180 GOSUB6000
190 EXPLODE
198 REM
```

```
200 REM   DESSIN DECOR
202 REM
210 GOSUB9000:GOSUB8400:CLS
220 PAPER4:INK0:K=2
230 D=0:X=37:Y=7:A=2.2:AM=2:NO=30:AX=0:CA=300:A1=2:B1$=" "
260 GOSUB2000
270 GOSUB5000
280 FORN=23TO26
290 PLOT1,N,"{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{{"
300 NEXT:PLOT1,6,"----------------------------------------"
998 REM
1000 REM   BOUCLE PRINCIPALE
1002 REM
1100 X$=KEY$:IFX$<>""THENAX=ASC(X$)
1105 IFCA<0THENAX=10
1110 IFAX>9ANDAX<12THENGOSUB2000
1120 IFX$<>" "THEN1130
1125 IFB1$=" "THENB1$=B$:Y1=Y:X1=X+1:AS=SCRN(X1,22):MU=MU-1
1130 ONDGOTO3000:REM  Avion detruit?
1140 GOSUB4000  :REM  Dessin decor
1150 PLOTX,Y,R$
1160 IFX>2THENX=X-1ELSEX=36:CA=CA-30:CO=CO+10:GOSUB5000
1165 D=(X=A1)*(Y=AL)
1170 PLOTX,Y,V$
1180 PLOTAM-1,AL,0:PLOTAM,AL,"!"
1185 ONKGOTO1200,1190
1190 PLAY7,7,6,NO:K=1
1197 REM
1198 REM   Explosion bombe
1199 REM
1200 IFB1$=" "ORMU=0THEN1100
1210 Y1=Y1+1
1220 IFY1<23THEN1280
1225 PLOT0,22,7:EXPLODE:WAIT5:PLOT0,22,0
1230 PLOTX1,22," ":K=2
1231 REM
1232 REM   Cible touchee
1233 REM
1235 IFAS=37THENCO=CO-50:AS=0:GOTO1250
1237 IFAS=65THENCO=CO-100:AS=0:CA=CA+30:GOTO1250
1240 IFX1<>A1THEN1270
1250 PAPER7:INK0:EXPLODE:WAIT5:PAPER0:INK7:CO=CO+200:PAPER4:INK0:K=
2
1260 GOSUB5000
1270 B1$=" ":GOTO1100
1280 PLOTX1,Y1-1," "
1290 PLOTX1,Y1,B1$
1300 GOTO1100
1998 REM
2000 REM   Deplacements Avion
2002 REM
2030 IFAX=11ANDY>7THENPLOTX,Y,R$:Y=Y-1:NO=NO+1:K=2:GOTO2100
```

```
2040 IFAX<>10THEN2100
2050 IFY<21THENPLOTX,Y,R$:Y=Y+1:NO=NO-1:K=2ELSED=1
2100 PLOTX,Y,V$:RETURN
2998 REM
3000 REM   DESTRUCTION AVION
3002 REM
3010 FORN=0TO5
3015 IFCO>DEEK(#400)THENDOKE#400,CO
3020 PAPERRND(1)*7:INKRND(1)*7:EXPLODE:WAITRND(1)*10
3030 WAIT2:NEXTN
3035 IFVA=1THEN3040
3036 VA=VA-1:GOSUB8000:WAIT250:CLS:GOTO220
3040 PAPER7:INK0:GOSUB5000
3050 POKE616,7:PRINTCHR$(4)
3060 PRINTE$"N           GAME  OVER":PRINTCHR$(4);
3070 RUN
3998 REM
4000 REM   DESSIN DECOR
4002 REM
4010 PLOT1,22,9
4020 A$=CHR$(RND(1)*60+32)
4030 PLOTA,22,A$:PLOTA,AL," "
4031 YL=12:IFY>12THENYL=Y
4032 AL=INT(RND(1)*R1+YL-(R1-1))
4033 AM=A1
4035 PLOTA-1,AL,7:PLOTA,AL,"*"
4040 A=A+.8:A1=INT(A):IFA1>37THENA=2.2:A1=2
4050 RETURN
4998 REM
5000 REM   COMPTEUR
5002 REM
5010 POKE616,1:PRINTCHR$(4)
5015 IFCO/1000>=MITHENVA=VA+1:MU=MU+8:MI=MI+1
5020 PRINTE$"JFUEL:"CA"  BOMBES:"MU"  SCORE:"CO:PRINTCHR$(4)
5022 POKE616,4:PRINT:PRINT"  HI SCORE:"DEEK(#400):PRINT"  ";
5023 FORN=1TOVA:PRINTV$" ";:NEXT
5030 RETURN
5998 REM
6000 REM   MESSAGE DE DEBUT
6002 REM
6005 IFDEEK(#400)<>21845THENFORN=1TO11:READZ,Q:NEXT:GOTO6150
6010 CLS:POKE616,3:PRINTCHR$(4)
6020 PRINTE$"E"E$"J   BVRP      PRESENTE:":POKE616,11:PRINT
6030 PRINTE$"N        NIGHT   BOMBER"
6040 POKE616,20:PRINTCHR$(4)
6050 PRINTE$"E     A  GAME  OF  SKILL !"
6055 O=3
6060 FORN=1TO11
6070 READA,B
6075 MUSIC2,O,A,12
6080 PLAY3,0,1,2000
6090 WAITB
6095 PLAY0,0,0,0
```

```
6100 NEXTN
6110 IFO=2THEN6120ELSEO=2:RESTORE:GOTO6060
6120 DOKE#400,0:GOSUB8200
6150 RETURN
6200 DATA5,30,5,30,3,15,8,75,5,30
6210 DATA8,40,10,30,7,20,5,30,3,30,5,90
7998 REM
8000 REM   CRASH
8002 REM
8010 POKE616,13:PRINTCHR$(4)
8020 PRINTE$"G            "E$"NCRASH!"E$"H"E$"@"CHR$(4)
8030 RETURN
8198 REM
8200 REM   TEXTE DEBUT
8202 REM
8210 CLS
8220 PRINT:PRINT
8230 PRINTE$"GVOTRE BOMBARDIER EST EN MISSION DE":PRINT
8240 PRINTE$"GNUIT AU DESSUS DU TERRITOIRE ENNEMI":PRINT
8250 PRINT:PRINTE$"G     ATTENTION A LA D.C.A!"
8260 RETURN
8398 REM
8400 REM   EXPLICATIONS
8402 REM
8405 POKE616,10:PRINT
8410 PRINTE1$"V"E1$"A         REGLES  DU  JEU          "
8420 PRINTSPC(38);:PRINT"  ^ @ # : HAUT/BAS/TOUT DROIT  "
8430 PRINT"  ESPACE: LARGUE UNE BOMBE  ":PRINTSPC(38);
8440 PRINTE$"I"E$"H:"E$"F150 POINTS
8450 PRINTE$"IA"E$"H:"E$"F100 PTS & 30 LITRES DE FUEL"
8460 PRINT"CURS.:"E$"F200 POINTS":PRINTSPC(38);
8500 PRINT
8510 PRINTE1$"VCHAQUE PASSAGE CONSOMME 30 LITRES    "
8520 PRINTE1$"VTOUS LES 1000 PTS: 8 BOMBES & 1 AVION ";
8530 PRINT:PRINT:PRINT:PRINTE$"L"E1$"FTAPER <1> OU <2> SELON NIVEAU
     ";
8540 X$=KEY$:IFX$="1"ORX$="2"THENR1=4-VAL(X$):RETURN
8550 GOTO8540
8999 REM
9000 REM   REDEFINITION CARACTERES
9002 REM
9100 READAD:IFAD=0THEN9700
9110 FORN=0TO7
9120 READDA:POKEAD+N,DA
9130 NEXT
9140 GOTO9100
9500 DATA 47400,0,8,8,8,8,8,28,54
9510 DATA 46904,0,0,0,2,28,2,0,0
9520 DATA 47928,4,37,21,14,63,14,21,37
9530 DATA 47072,6,14,31,62,31,12,0,0
9540 DATA 47624,12,30,63,63,30,12,18,51
9550 DATA 46840,0,12,63,56,31,6,4,0
9560 DATA 46816,3,7,63,30,49,0,0,0
```

```
9570 DATA 47064,42,21,42,21,42,21,42,21
9580 DATA 46592,8,8,8,8,42,28,8,0
9590 DATA 46360,0,8,16,63,16,8,0,0
9690 DATA0
9699 REM
9700 REM   INITIALISATIONS
9702 REM
9710 READ X,Y,A,AL,NO,AM,A1,VA,MU,CA,MI
9720 READV$,B$,F$,R$,B1$
9799 RETURN
9800 DATA 37,7,2.2,20,30,2,2,3,50,300,1
9810 DATA _\,g,%,"  ","  "
```

Explications du programme

Plusieurs caractères spéciaux sont créés par le jeu :

"%",	jeu graphique	: Fusée
"A",	jeu graphique	: Réservoir
"g",	jeu normal	: Bombe
"g",	jeu graphique	: Explosion au sol
"£",	jeu normal	: Avant de l'avion
"\",	jeu normal	: Arrière de l'avion
"|",	jeu normal	: Nuage de fumée (explosion de D.C.A.)

On remarque que plusieurs caractères sont placés dans le jeu graphique, car toute la ligne où défile le décor composé de mosaïques graphiques est précédée d'un attribut "deuxième jeu". Ainsi, le "g" qui représente une bombe dans le jeu normal (tant que la bombe est en l'air) se transformera instantanément en explosion une fois arrivé au sol. De plus, l'effet d'explosion est renforcé par la modification de l'attribut de couleur de la première ligne pendant un court instant, qui passe du noir au blanc quand aucune cible n'est touchée. En cas de coup réussi, c'est tout l'écran qui change de couleur accompagné par une succession de "EXPLODE" du plus bel effet...

Étant donné leur profusion, nous avons jugé utile d'ajouter une liste détaillée des variables employées :

Chaînes

B$: Bombe ; caractère "g".
B1$: Contient un espace en temps normal, et B$ lorsque la bombe est en chute.

V$: Avion; deux caractères.
R$: Contient deux espaces, pour effacer l'avion lorsqu'il se déplace.
E$ Caractère ESCAPE — CHR$(27).

Coordonnées

A : Abscisse du curseur.
AM : Mise en mémoire de l'abscisse curseur.
A1 : Abscisse de la D.C.A.
AL : Altitude de la D.C.A. (ordonnée).
X,Y : Abscisse et ordonnée de l'avion.
X1,Y1 : Abscisse et ordonnée de la bombe.

Compteurs

CO : Nombre de points (score).
CA : Compteur carburant.
MU : Compteur munitions.
VA : Nombre d'avions restants.

Drapeaux

D : Égal à 0 si tout est normal, à 1 si l'avion est touché.
MI : Nombre de milliers de points gagnés, détermine l'acquisition d'avions supplémentaires.

Divers

R1 : Niveau du jeu (1 ou 2).
NO : Paramètre dans une instruction PLAY, détermine la hauteur du son en fonction de l'altitude de l'avion.
AS : Code du caractère se trouvant au point d'impact de la bombe; si celui-ci correspond à une fusée ou un réservoir, les compteurs sont incrémentés en conséquence.
AX : Code ASCII de la touche pressée.

— 180: Appel d'un sous-programme jouant un petit thème musical, au premier démarrage du programme (6000).

— 210: Appel des sous-programmes d'initialisation et création de caractères spéciaux (9000), et d'affichage des explications (8000).

— 220-230: Initialisation de certaines variables. La routine située en 9000 n'étant appelée qu'à la première exécution du programme,

certaines variables doivent être initialisées localement quand on veut rejouer.

— 280-300 : Remplit le bas de l'écran avec une zone grisée (le "sol").

— 1000-1190 : Boucle principale, gérant les déplacements de l'avion, le son qu'il fait et les mouvements de la bombe si elle a été lâchée.

— 1200-1230 : Génère les son et image correspondant à l'explosion de la bombe au sol.

— 1235-1300 : Détermine si une cible a été touchée, en testant le code du caractère situé à l'abscisse de la bombe. Si c'est le cas, génère les son et image appropriés et incrémente les compteurs.

— 2000-2100 : Efface l'avion de son ancienne position, et le redessine à la nouvelle.

— 3000-3070 : Le branchement s'effectue vers cette routine si le drapeau de destruction (D) est à 1, résultat de la concordance entre l'altitude de la D.C.A. (AL) et l'ordonnée de l'avion (Y). Dans ce cas, l'écran prend une succession de couleurs aléatoires accompagnées d'explosions. S'il ne reste plus d'avions, le message "GAME OVER" s'affiche et le programme est relancé.

— 4000-4050 : Dessine des mosaïques graphiques aléatoires au niveau du sol, en parsemant de ci, de là quelques cibles tout aussi aléatoires...

— 5000-5030 : Réactualise le compteur à chaque tour d'écran de l'avion ; se charge aussi de l'attribution d'avions supplémentaires tous les 1000 points. Si vous voulez changer cette valeur, il suffit de modifier la ligne 5015.

— 6000-6210 : Joue la musique "de générique" à la première exécution du programme. Cette partie peut être omise, en veillant à supprimer le "GOSUB 6000" de la ligne 180.

— 8000-8030 : Affiche le message "CRASH" si l'avion est détruit et qu'il en reste encore.

— 8400-8550 : Affichage des explications (points donnés par chaque cible, commandes, etc.) en début de programme.

— 9000-9810 : Création des caractères spéciaux dont les données, contenues dans des DATA, sont précédées de l'adresse de début du caractère ; initialisation des principales variables.

Appendices

Appendice 1 – Le jeu de caractères ASCII

Code		**Caractère**	**Code**		**Caractère**
Décimal	*Hexa*		*Décimal*	*Hexa*	
32	20	Espace	42	2A	*
33	21	!	43	2B	+
34	22	"	44	2C	,
35	23	#	45	2D	–
36	24	$	46	2E	.
37	25	%	47	2F	/
38	26	&	48	30	0
39	27	'	49	31	1
40	28	(	50	32	2
41	29	)	51	33	3

Code Décimal	Code Hexa	Caractère	Code Décimal	Code Hexa	Caractère
52	34	4	89	59	Y
53	35	5	90	5A	Z
54	36	6	91	5B	[
55	37	7	92	5C	\
56	38	8	93	5D	]
57	39	9	94	5E	↑
58	3A	:	95	5F	£
59	3B	;	96	60	©
60	3C	<	97	61	a
61	3D	=	98	62	b
62	3E	>	99	63	c
63	3F	?	100	64	d
64	40	@	101	65	e
65	41	A	102	66	f
66	42	B	103	67	g
67	43	C	104	68	h
68	44	D	105	69	i
69	45	E	106	6A	j
70	46	F	107	6B	k
71	47	G	108	6C	l
72	48	H	109	6D	m
73	49	I	110	6E	n
74	4A	J	111	6F	o
75	4B	K	112	70	p
76	4C	L	113	71	q
77	4D	M	114	72	r
78	4E	N	115	73	s
79	4F	O	116	74	t
80	50	P	117	75	u
81	51	Q	118	76	v
82	52	R	119	77	w
83	53	S	120	78	x
84	54	T	121	79	y
85	55	U	122	7A	z
86	56	V	123	7B	}
87	57	W	124	7C	¦
88	58	X	125	7D	{

Appendice 2 – Index des adresses mémoire

Adresse	*Fonction*	*Page(s)*
Pointeurs		
#9A,#9B (154,155)	Début zone programme	37, 44
#9C,#9D (156,157)	Fin prog./début var. simples	37, 39, 44, 56
#9E,#9F (158,159)	Fin var.simples/début tableaux	39, 42
#A0,#A1 (160,161)	Fin tableaux/début chaînes	40, 42
#A2,#A3 (162,163)	Adresse la plus basse dans la zone chaîne	42
#A4,#A5 (164,165)	Début de la dernière chaîne entrée ..	42
#A6,#A7 (166,167)	HIMEM	43, 12
#B0,#B1 (176,177)	LOMEM	43
#B8,#B9 (184,185)	Adresse dernière variable entrée ...	43
L'Écran		
#12,#13 (18,19)	Adresse début de la ligne du curseur	47, 79
#268 (616)	Numéro de la ligne du curseur	48, 27, 79
#269 (617)	Numéro de la colonne du curseur ...	48, 27, 79
#26A (618)	Modes curseur et clavier	48, 70
#26B (619)	Couleur du fond, pour tout l'écran ..	50, 80
#26C (620)	Couleur affichage, pour tout l'écran .	50, 80
#26D,26E (621,622)	Adresse début de la première ligne de l'écran accessible au curseur	50, 77
#26F (623)	Numéro de la dernière ligne de l'écran accessible au curseur	50, 78
#2C0 (704)	Pointeur TEXT/HIRES	51
Le clavier		
#208 (520)	Code de la dernière touche frappée .	46
#209 (521)	Pointeur Normal/CTRL/SHIFT	46
#20C (524)	Pointeur Majuscules/Minuscules ...	46
#20E (526)	Déclenchement de l'autorepeat	46
#306,307 (774,775)	Vitesse de l'autorepeat	47

Adresse	*Fonction*	*Page(s)*
Le magnétocassette		
#35-#46 (53-70)	Nom du fichier à transférer	57
#5F,#60 (95,96)	Adresse début de zone à transférer .	57
#61,#62 (97,98)	Adresse fin de la zone à transférer ..	57
#63 (99)	Pointeur AUTO/Normal	57
#64 (100)	Pointeur Prog.Basic/Zone mémoire .	57
#67 (103)	Pointeur vitesse de transfert	57
Les zones-mémoire		
#400-#4FF (1024-1279)	Libre	51, 56
#500 (1280)	Début programme	51, 38
#501 (1281)	1er octet du programme	38, 39
#503 (1283)	Fin zone programme (à vide)	37, 40, 41
#B400 (46080)	Début jeu ASCII	57, 82, 52
#B7FF (47103)	Fin jeu ASCII	82, 52
#B800 (47104)	Début jeu graphique	82, 52
#BB57 (47959)	Fin jeu graphique	82, 52
#BB80 (48000)	Début écran texte	52, 57, 62
#BFE0 (49120)	Fin écran texte	57, 62
#A000 (40960)	Début écran haute-résolution	106, 107
#BF3F (48959)	Fin écran haute-résolution	106, 107
#BF68 (49000)	Début lignes texte en HIRES	106, 107
#BFE0 (49120)	Fin lignes texte en HIRES	106, 107
Routines de la ROM (Appelées par CALL)		
#E4A8 (58536)	Chargement fichier (CLOAD)	58
#E57B (58747)	Sauvegarde fichier (CSAVE)	58
#ED01 (59082)	Blocage du clavier	53, 58
#E804 (59396)	Déblocage du clavier	53, 58
#F84A (63562)	RESET total	52
#F882 (63618)	RESET partiel	52

Appendice 3 – Messages d'erreur

Si une commande, donnée ou syntaxe inacceptable est validée, ORIC répondra par un des messages ci-dessous suivis du mot "ERROR"

CAN'T CONTINUE

Réponse à l'instruction CONT, si le programme a été modifié pendant l'interruption.

DISP TYPE MISMATCH

Affiché si des instructions graphiques sont utilisées en mode texte, ou si PLOT est utilisé en HIRES.

DIVISION BY ZERO

Refus de diviser un nombre par zéro.

FORMULA TOO COMPLEX

trop de IF — THEN imbriqués sur une même ligne.

ILLEGAL DIRECT

Tentative d'utiliser en mode direct des commandes ne fonctionnant que dans un programme (GET, INPUT etc.).

ILLEGAL QUANTITY

Une valeur hors limites a été donnée comme argument à une instruction ou fonction (par ex. PRINT PEEK(300).

NEXT WITHOUT FOR

Une commande NEXT est rencontrée sans que la boucle ait débuté par FOR.

OUT OF DATA

Tentative d'utiliser READ alors que les DATA ont déjà été lus en entier.

OUT OF MEMORY

Plus de mémoire, ou bien plus de 16 boucles du type FOR-NEXT, REPEAT-UNTIL imbriquées.

OVERFLOW

Les calculs ont dépassé le nombre $1.7*10^{38}$.

REDIM'D ARRAY

Réponse à une tentative de redimensionner un tableau déjà déclaré dans un DIM.

RETURN WITHOUT GOSUB

Un RETURN est rencontré sans être préalablement passé par un GOSUB. Peut aussi être provoqué par un POP mal utilisé.

STRING TOO LONG

Tentative d'attribuer plus de 255 caractères à une chaîne.

BAD SUBSCRIPT

Un élément de tableau est appelé en utilisant un indice hors des limites déclarées dans DIM, lorsque le tableau a été dimensionné.

SYNTAX ERROR

Une erreur de syntaxe a été rencontrée dans une commande directe ou une ligne de programme.

TYPE MISMATCH

Discordance entre un type de variable et les données qu'on essaie de lui attribuer (par ex. A="ABCD"), ou entre une commande et l'argument qu'on lui donné (ex. SIN(A$)).

UNDEF'D STATEMENT

Un numéro de ligne inexistant a été fourni comme argument à GOTO, THEN ou GOSUB.

UNDEF'D FUNCTION

Tentative d'utiliser une fonction avec FN, sans l'avoir préalablement définie par DEF FN.

REDO FROM START

Des données incorrectes ont été fournies à un INPUT. Seule erreur ne repassant pas en mode direct, l'INPUT est reexécuté.

BAD UNTIL

Un UNTIL est rencontré, sans que la boucle ait commencé par REPEAT.

Index alphabétique

A

Adressage curseur 79
Adresses :
  Écran 62
  Mémoire 34
  De retour 45
APPLICATION 6
AND 5, 13, 18
ASCII 23, 62
Arguments 5
Arrondi exact 30
Attributs d'écran 50, 68, 71, 80
AUTO 8
Autorépétition (du clavier) . 46

B

Basic 5
Basic Microsoft 6
Baud 8
Bits 33, 82, 112
BREAK 64
Bruit blanc 131, 133
Buffer de ligne 66, 67

C

Canal 131, 132
CALL 8, 58, 53
Chaînes de caractères 3
CHAR 110
CHR$ 53, 63, 64
CIRCLE 110
Clavier 11, 21, 46
CLEAR 39
Clignotants (caractères) ... 69, 72
CLOAD 9, 55
CLS 2, 65
Codes de contrôle 64
Colonnes réservées 63
Concaténation (de chaînes) 4
Constantes :
  De chaîne 3
  Hexadécimales 3
Conversion (degrés/radians) 20, 30
CONTROL (CTRL) 46, 48, 63, 67
Coordonnées (du curseur) . 62
Coordonnées 108
Couleurs 14, 15, 68, 71, 109

CSAVE 55, 8
CURMOV 110
CURSET 109
Curseur graphique 108

D

D.A.O. 119
DATA 25
DEEK 20
DEF FN 9
DEF USR 10
DEL 65, 66
DIM 10, 42
Dimensions (d'un tableau) . 3
Division entière 32
Division modulo 31
DOKE 11
Double hauteur 69
DRAW 110
Durée (d'un son) 133

E

Écran 47, 61, 62
EDIT 11, 66
Éditeur de ligne 66
Ellipse 116
Enveloppe (d'un son) 133
ESCAPE 68, 69, 71, 72
EXEMPLE 7
EXPLODE 131
Espaces 68

F

FALSE 19
FILL 111
Flèches de déplacement .. 64
FN 20
Fonctions 2, 19, 30, 114
Formatage d'un nombre .. 27
FRE 20

G

GET 11
GOSUB 15
GRAB 12
Graphique (haute résolution) 105

H

Hardcopy graphique 117
Hardcopy texte 70
Hexadécimal 3, 21
HEX$ 21
HIMEM 12, 36, 43
HIRES 109

I

IF-THEN 19
IF... THEN... ELSE 13
Imprimante 53
Indices 4, 6
INK 14, 15
INKEY$ 22
INPUT 7
INPUT$ 31
INSTR 31
Instructions 2, 8, 26, 109, 131
INT 6, 30
Interpréteur Basic 2
Interface parallèle 53

J

Jeu standard 80, 81
Jeu semi-graphique 80, 81
Jeux de caractères 80, 81, 52

K

KEY$ 21

L

Langage machine 76
LET 2
Ligne réservée 63
Lignes de texte 107
LIST 14
LLIST 53
LOMEM 43
LORES 80
LPRINT 53

M

Magnétophone K7 54
Memory-map 34, 107
MID$ 24, 26
Minuscules 46
Modes :
  Direct 5
  Programme 5
Mosaïques graphiques 81, 82
Mots-clé 2
Mémoire :
  Morte (MEM) 2, 52
  Organisation 33
  Utilisateur 51
MUSIC 132

N

NEW 39, 44
NOT 5
Notation scientifique 3
Note (de musique) 132

O

Octave 132
Octet 6, 33, 82
ON... GOSUB 15
ON... GOTO 15
Opérateurs:
  Logiques 5
  Arithmétiques 4
  Relationnels 4
Opérandes 3
OR 5

P

PAPER 15
PATTERN 113
PEEK 76
Périphériques 52
Pile 44
PING 131
PLAY 133
PLOT 16, 71
Poids (d'un bit) 34, 82, 112
POINT 114
Pointeurs 36, 37
POKE 78, 76
Ponctuations 7
POP 17
POS 22
Positions d'affichage 61, 62
PRINT AT 27, 79
PRINT 63
PRINT USING 27
PULL 17

R

RAM 34
RAMTOP 12
Rectangle 116
Redimensionner (un tableau) 42
RELEASE 18
REMARQUES 6
RENUM 29
Renumérotation (de lignes) 29
REPEAT — UNTIL 18
RESET 8, 52
RETURN 17, 45, 67
RND 23
ROM 2, 52
RUN 5

S

Sauvegarde:
  D'écran 57
  De variables sur K7 58
SCRN 23
SHIFT 46
SHOOT 131
Son 130
Sons pré-programmés 131
SOUND 132
SPACE$ 32
SPC 23
Stockage de:
  Chaînes 41
  L'écran graphique 106
  Lignes de programme .. 38
  Pointeurs de tableaux de chaînes 41
  Tableaux numériques ... 41
STR$ 71
STRING$ 32
SWAP 30
SYNTAXE 6
Système d'exploitation de cassettes 54

T

TAB 24
Tableaux:
  De chaîne 3, 4
  De variables entières ... 6
TEXT 80, 109
Trait (du dessin) 113
Transfert (de zones-mémoire) 54, 55
TROFF 18
TRON 18
TRUE 19

U

USR 24

V

Variable:
  De chaîne 3, 41
  Entières 3, 40
  Indicées 4
  Simples 3, 39
  Système 45
Vidéo inverse 16, 71

W

WAIT 19
WHILE-WEND 18

Z

ZAP 131
Zones:
  Mémoire 36
  Programme 37, 38
  Variables 39, 41

Achevé d'imprimer
par **Cid** *éditions*

Dépôt légal : mars 1984
N° d'Éditeur : 4056

Ce livre explore toutes les possibilités de l'Oric-1 dans le domaine de la programmation avancée. Une étude complète des adresses mémoire et des variables système ouvre la voie à des extensions du Basic, telles que la gestion complète de l'écran, la sauvegarde de variables sur cassette, la hardcopy etc...

Son et graphique y sont très largement traités, et font l'objet de nombreux programmes dont un jeu d'arcades en temps réel.

Trois grands logiciels d'applications sont inclus :

— Un générateur de caractères très performant et simple à utiliser.

— Un traitement de textes autorisant la saisie et sauvegarde de vos documents sur cassette, ainsi que leur impression et l'utilisation des jeux de caractères que vous avez dessinés (minuscules accentuées, alphabet grec etc...).

— Un programme de dessin assisté par ordinateur en couleurs, comportant de nombreuses formes géométriques pré-programmées et incluant la sauvegarde/relecture des écrans graphiques.

EYROLLES

8209

TOUT SAVOIR SUR ORIC

R. POLITIS
B. VANRYB